



SHARP CORPORATION

mz-80k



SYSTEM PROGRAM

ASSEMBLER	SP-2101
TEXT EDITOR	SP-2201
RELOCATABLE LOADER	SP-2301
SYMBOLIC DEBUGGER	SP-2401
EDITOR-ASSEMBLER(OPTION)	SP-2102
	SP-2202

MZ-80系システムは、クリーンコンピュータを設計思想として開発されたものです。製品定格および命令仕様は、将来のバージョンアップで予告なしに変更、追加、削除されることがあります。そのため、提供されるモニター（ROMで提供）、システム・プログラム（カセットテープなどで提供）のバージョンナンバーには特にご注意ください。

記載事項について間違いはないものと信じておりますが、本資料は "SYSTEM PROGRAM" をご理解いただくため作製したものですので、バージョンアップによる内容の相違、誤植、記載もれなどからおこる問題点に関しては、原則的に責務は負いかねます。

なお、本書の作製は、モニター SP-1002, ASSEMBLER SP-2101, TEXT EDITOR SP-2201, RELOCATABLE LOADER SP-2301, SYMBOLIC DEBUGGER SP-2401 にもとづいております。

目次

第 1 章	SYSTEM PROGRAM 概要	1
第 2 章	ASSEMBLER SP-2101	7
2-1	アセンブラ概要	9
2-2	アセンブラ言語規約	10
2-3	アセンブルリストとアセンブラメッセージ	14
2-4	アセンブラ擬似命令の考え方	17
2-5	アセンブラの各PASS	24
第 3 章	TEXT EDITOR SP-2201	31
3-1	テキストエディタ概要	33
3-2	キャラクタポイント (CP) とデリミタ	35
3-3	テキストエディタコマンド	36
3-4	EDITOR-ASSEMBLER (オプション) の使い方	50
第 4 章	RELOCATABLE LOADER SP-2301	51
4-1	リロケータブルローダ概要	53
4-2	ローダのバイアスとアドレスの考え方	55
4-3	リロケータブルローダコマンド	58
第 5 章	SYMBOLIC DEBUGGER SP-2401	65
5-1	シンボリックデバッガ概要	67
5-2	ブレークポイントの考え方	68
5-3	シンボリックデバッガコマンド	69
5-4	セーブファイルとBASICテキストとのリンク方法	87
5-5	各システムプログラム間のファイルの考え方	88
第 6 章	アセンブラプログラミング例題集	89
	(1) 接近して来る四角形 (2) データの分類 (3) デジタル時計	
	(4) 16進の乗算 (5) 16進データの表示 (6) 16進データの入力	
	(7) メモリダンプ (8) メモリライト (9) メモリリストダンプ	
	(10) オルガン	
第 7 章	MZ-80K システムコントロール 資料と解説	127
7-1	アスキーコード表	129
7-2	モニタサブルーチンの使い方	130
7-3	ディスプレイコード表	133
7-4	割込みを使用する上での注意	134
7-5	E000H 番地内の考え方	135
7-6	メモリマップドI-Oチャート	136
7-7	ハードウェアリセットの考え方	137
7-8	参考文献	138
附 録	SYSTEM PROGRAM COMMAND 一覧表	

mz-80k



第 1 章

SYSTEM PROGRAM

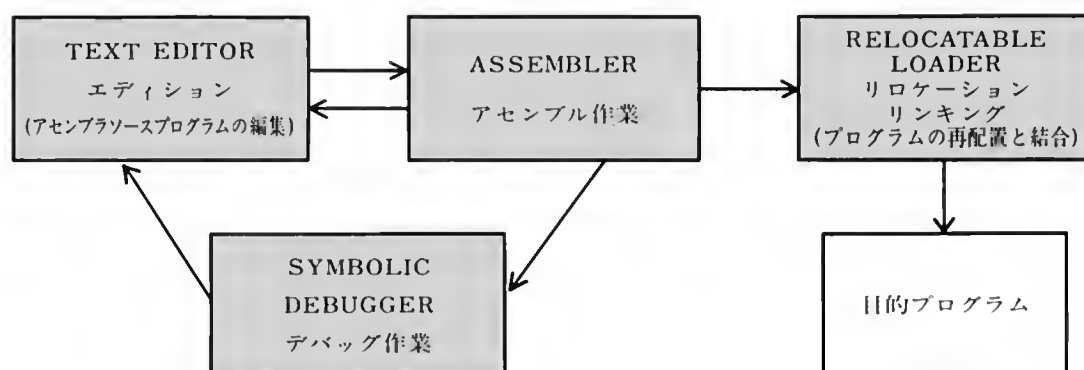
概要

mz-80k



SYSTEM PROGRAMは、ASSEMBLER、TEXT EDITOR、RELOCATABLE LOADER、SYMBOLIC DEBUGGERの各システムプログラムから構成される、Z80アセンブラプログラミングを行うためのソフトウェアセットである。アセンブラプログラミングは、コンピュータシステムの中央処理装置(CPU)が実行することのできる命令群の1つ1つを表現した記号、すなわちアセンブリ言語によってプログラミングを行うものであり、システムを、いわばその素材としての基本機能から追求するものであるといえることができる。CPUレベルでのプログラミングの方法には、機械語コードを命令語としてじかに取り扱う方法も行われるが、その場合に生ずるいろいろな困難な点、たとえば命令が直感的に捉えにくいこと、アドレッシングが相対化できないこと、変数の概念を活用できないことなどについて、アセンブラプログラミングでは有機的に機能が高められ、困難が取り除かれている。すなわち、プログラマは、CPUインストラクションセットのニモニックを用いてプログラムを記述し、さらにその命令中で参照するデータ、アドレス等の値をプログラマが任意に選んだシンボルを使って表現することができる。アセンブリ言語によるプログラムをCPUが実行できる機械語へ変換する作業は、アセンブラおよびローダが機械的に行うのである。機械語コードを命令として直接用いるプログラミングを行うためにMACHINE LANGUAGE (SP-2001) がサポートされている。このソフトは、CPU動作についての学習を主眼としたものであるが、SYSTEM PROGRAMは、その上位に位置づけられ、機動性、開発性が十分に盛られているのである。

アセンブリ言語を用いたアセンブラプログラミング(ソースプログラムの作成)から、目的プログラムの作成までをアセンブル過程と呼ぶことにすると、アセンブル過程には、プログラムのアセンブル、リロケーション、リンキング、デバッグ、エディション等の幾つかの過程が含まれており、各過程の実行はシステムプログラムの各ソフトがそれぞれ担当する。



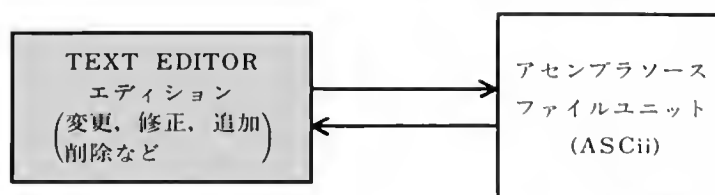
アセンブル過程概念図

上図は、SYSTEM PROGRAMによるアセンブル過程の概念を示している。目的プログラムの作成までに幾つかの段階的な過程を経ていることがわかる。プログラムの作成段階は、上図の左の部分、すなわち、エディション、アセンブル作業、デバッグ作業を繰り返すことによって進められる。最初に、TEXT EDITORによってアセンブラソースプログラムを作成し、プログラムファイルをASSEMBLERに入力する。そこでアセンブラソースプログラムのアセンブル作業が行われるが、その際プログラム中にエラーが検出されれば、アセンブラメッセージとしてエラー表示が得られるので、エディタへ戻ってソースプログラムの変更、修正作業を行う。アセンブラ規約上のエラーが排除されたら、リロケータブルバイナリファイルを出力して、SYMBOLIC DEBUGGERに入力する。デバッグは、プログラムを、そのリンクエリア内に実行可能な形で構成して、実際にプログラムを走らせることができる。その際、プログラマは、プログラム中に適宜ブレークポイントを置いて、プログラムの実行を中断、継続させたり、CPUレジスタの内容、メモリの内容を表示させ変更するなどの、いわゆるデバッグ操作を行うことができる。このデバッグ過程によって、プログラムの構成上の誤りなどを見つけ出したら、やはりエディタへ戻って、ソースプログラムの編集をしないおすことになる。こうして、アセンブラソースプログラムの作成が終了したら、アセンブラ出力のリロケータブルバイナリファイル(複数のファイルであってよい)をRELOCATABLE LOADERに入力して、目的とする実行形式をもつ機械語プログラムを得る。

SYSTEM PROGRAMの各ソフトウェアについて、それがアセンブル過程においてになう役割、機能、出力ファイルの性質などの点を順に概説する。

TEXT EDITOR (SP-2201) は、アセンブラソースプログラムユニットを作成、または編集するシステムプログラムである。ここで、アセンブラソースプログラムユニットと言うのは、作成するプログラムの部分的な単位を示すものであり、通常、目的プログラムは、幾つかのプログラムユニット（たとえばメインプログラム、サブルーチンプログラム、データ群などのユニット）をリンクして構成される。アセンブラソースプログラムユニットは、アセンブリ語命令文、ラベルシンボル、コメント文、セパレータ、行の区切りなどの要素から成り、それらは、ASCIIコードによってコード化される。エディタ出力として得られるアセンブラソースファイルは、そうしてエディットバッファ内にコード化されたアセンブラソースプログラムユニットをそのままファイル上へ移したものである。

アセンブラソースファイルは、プログラムの変更、修正、追加のために、エディットバッファへローディングして戻ることができる。こうしてアセンブラソースファイルに必要な手を加えて行き、プログラムを完成するのである。



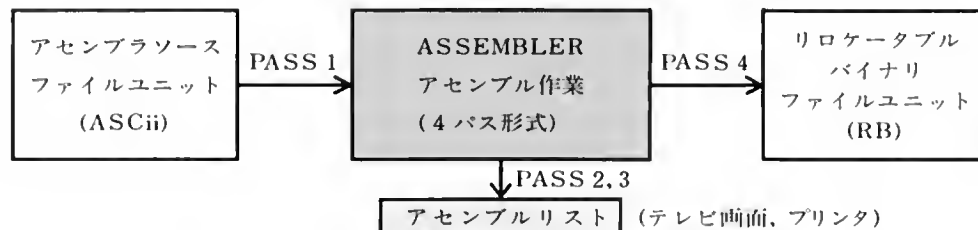
TEXT EDITORによるエディション過程

アセンブラソースプログラムユニットの変更、追加等の編集作業を行うのに、エディットバッファ内に置かれたキャラクタポインタ (character pointer) というポインタを用いる。これは、BASICテキストの編集で使われるカーソルに相当するような役割を果たす。

ASSEMBLER (SP-2101) は、SYSTEM PROGRAMの中核となる働きをもつ。すなわち、TEXT EDITORによって出力された、アセンブラソースプログラムユニットを読み込み、アセンブラ規約に従ってアセンブル作業を実行する。ASSEMBLERは出力として、アセンブルリストの表示 (テレビ画面上および、プリンタ上) と、リロケートブルバイナリファイルを得る、リロケートブル形式のマクロアセンブラである。

入力アセンブラソースファイルは、ラベル、オペレーションのニモニク記号、擬似命令、注釈文、エンド文から構成され、それらはアセンブラ規約に従う文法で記述されなければならないが、もしその中に文法上の誤り等が検出されると、エラーメッセージがアセンブルリスト上へ表示される。

出力のリロケートブルバイナリファイルは、ソースファイルと目的ファイル (アブソリュートバイナリファイル) の間で、プログラムを再配置 (relocation) およびリンク (linking) が可能な状態として構成するファイルである。即ち、アセンブル作業は、アセンブリ言語の機械語コード化を行うものであるが、オペランドのアドレス形式および、リンク形式については未定義状態として、最終的にローダによって絶対アドレスが決定されるようになっている。ここにアセンブラプログラミングのグローバルな、結合、拡張性が図られている。



ASSEMBLERによるアセンブル過程

ASSEMBLERは4パス形式をとっている。PASS 1はソースファイルの読み込みによるシンボルテーブルの作成過程、PASS 2, PASS 3はそれぞれテレビ画面およびプリンタ上へアセンブルリストをタイプアウトする過程、そしてPASS 4は、リロケートブルバイナリファイルの出力過程となっている。

SYMBOLIC DEBUGGER (SP-2401) は、プログラムのデバッグを行うためのシステムプログラムである。SYMBOLIC DEBUGGER は ASSEMBLER 出力の、1 つのリロケートブルバイナリファイル、あるいは複数のリロケートブルバイナリファイルをリンクエリア内に、即時実行形プログラムとしてリンクングロードを行い、プログラムを実際に走らせることによってデバッグ作業を行うものである。

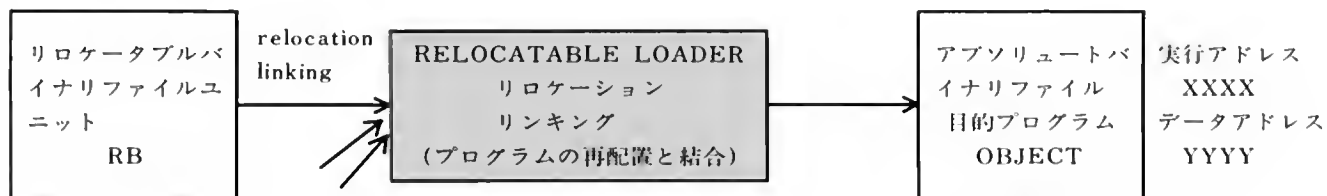
デバッグ作業は、リンクエリア内に構成されたアブソリュートバイナリの形式をメモリダンプによって調べたり、特にプログラム中の任意の箇所に、プログラム実行を中断させるブレイクポイント (break point) を置いて、CPU 内部レジスタの内容の変化を調べたり変更したりすることによって進めることができる。また、CPU 内部レジスタを、ある値に設定した状態で、任意の箇所からプログラムをスタートする (indicative start) こともできる。



SYMBOLIC DEBUGGER によるデバッグ過程

SYMBOLIC DEBUGGER における "SYMBOLIC" という概念は、デバッグ操作を行う際に指定するアドレス (たとえば、ブレイクポイントの設定箇所など) を、16進数表現の絶対アドレスで指定する以外に、ソースプログラム中で用いたエントリ宣言のなされているグローバルシンボルを用いて、相対的に指定することができるということを意味している。これによって、リロケートブルバイナリの段階での相対アドレス値やローディングの際指定したバイアス値などを覚えておく必要がなくなる。

RELOCATABLE LOADER (SP-2301) は、デバッグ作業が終了した時点で、最終的に目的プログラムを構成しファイル出力するためのシステムプログラムである。RELOCATABLE LOADER は複数のリロケートブルバイナリファイルユニットを、プログラマが指定する形式で再配置 (relocation) およびリンク (linking) を行いながらリンクエリア内に絶対形式 (アブソリュートバイナリ形式) でローディングして行き、それが終了したら、アブソリュートバイナリを目的ファイルへ、実行アドレスおよびデータアドレスを指定して出力する。こうして得られた目的ファイルは、それ自体で独立したプログラムと考えることができる。すなわち、モニタでローディングを行い、実行アドレスに指定したアドレスからプログラムを直ちに実行することができる。たとえば、1200 番地からスタートする "TINY BASIC" などを実現することが出来る。(MACHINE LANGUAGE ではこのような再配置や、実行アドレス、データアドレスの設定はできない。)



RELOCATABLE LOADER によるリロケーション、リンクング過程

RELOCATABLE LOADER は、プログラムのリロケーション、リンクングおよび、アブソリュートバイナリファイルの出力コマンドだけを持ち、実行コマンドあるいは編集コマンドは持たない。これは、1 つに、リンクエリア内に構成されるアブソリュートバイナリは、再配置とリンク作業のため一担メモリエリアを使うだけで通常実際のアドレスとは実行形式が異なるからであるし、またその必要もなく、できるだけサイズの大きな目的プログラムを構成することができるよう、デバッグツールとなるプログラムなどは一切持たず、リンクエリアとして使用できるエリアを拡げているからである。SYMBOLIC DEBUGGER のリロケーション、リンクングコマンドの持つ意味と異なる点である。

以上が SYSTEM PROGRAM としてサポートされる各プログラムの概要である。SYSTEM PROGRAM のアセンブル過程は、その各段階に於ていくつかのマクロ的な取り扱いができ、機械語によるプログラミングに伴う困難な側面が緩和されている。もちろんプログラミングのレベル自体は、やはりシステムに対してモザイク的な性格を持っていることに変わりはないが、そこにマクロ的な扱い易さだけでないもっと豊かな可能性が息づいているのである。

われわれのコンピュータシステム MZ-80 系は、"クリーンコンピュータシステム"という発想によって作られたパーソナルコンピュータである。"クリーンコンピュータ"という考え方は、一言で言うと、システムに、様々な発展性、拡張性を持たせるよう、機能を固定化しないということであり、具体的には、BASICプログラミング、機械語プログラミング、そしてこのアセンブラプログラミングなどを、それぞれに固有な特徴を充分発揮して行うことができるということである。すなわち、MZ-80 系システムは、メモリエリアの構成を、いつもソフト的にクリーンな状態とし、多様なシステムプログラム、拡張プログラムへの自由な対応が図られているのである。

SYSTEM PROGRAM のサポートは、MZ-80 系システムのこの"クリーンコンピュータ"という特質を更にもう一段高めた形で強調することになると考えられる。すなわち、SYSTEM PROGRAM は、文字通り MZ-80 系システムのシステム開発を行うものであり、BASIC、MACHINE LANGUAGE などと同列の新しいシステムプログラムを、ユーザの手で作り上げることができるからである。たとえば、ユーザの使用目的に合わせた、独自の BASIC インタプリタを作成したり、FORTRAN コンパイラ、PASCAL などの実現も不可能なことではないのである。

本書は、次章から各システムプログラムの解説、使い方の説明がなされたあと、例題集およびシステムコントロールに関する資料とから構成される。例題集は、10の例題からなり、システムプログラミングにとって基礎的な要素となるものがそれぞれ含まれている。程度が高いと思われる読者は、巻末に示した参考文献、あるいは、その他のアセンブラプログラミングの入門書で学習されるとよい。システムコントロールに関する資料は、MZ-80系システムのハードウェアとのつながりをまとめたものである。

第 2 章

ASSEMBLER

SP-2101

mz-80k



2-1 アセンブラ概要

ASSEMBLER SP-2101は、TEXT EDITOR SP-2201により作成、編集されたアセンブラソースファイルをアセンブルして、リロケータブルバイナリファイルを出力するリロケータブル形式のマクロアセンブラである。リロケータブルバイナリファイルは、ソースファイルと目的ファイル（アブソリュートバイナリファイル）の中間で、プログラムを、再配置（relocation）およびリンク（linking）が可能な状態として構成するファイルであり、アセンブラプログラミングによるグローバルな結合、拡張性が図られている。即ち参照アドレスは、最終的にローダで決定され、ラベルシンボルのグローバルエントリ宣言擬似命令ENTの使用によってローダで各プログラムユニットをリンクすることが可能である。

入力ファイルは、アセンブリ言語、即ち、ラベル、オペレーションのニモニック記号、擬似命令、注釈文、エンド文から構成され、それらはアセンブラ規約に従う文法で記述されなければならない。エディタによって編集されたソースプログラムはASCIIコードでコード化され、ファイル出力される。アセンブラは、このASCIIコードによるソースプログラムの構文解釈を行い、リロケータブルバイナリを作成するが、その際、シンボルアドレス（データ）の定義状態、文法の誤り等のメッセージの表示を行う。メッセージの表示は、テレビ画面上、またはオプションのプリンタ上でアSEMBルリスト中のメッセージ欄に行われる。

ASSEMBLER SP-2101は、4パス形式をとっている。アセンブラでのパス（PASS）とは、ソースファイルとしての1つのプログラムユニットあるいはデータユニットを初めから終わりまで1回システムが読み込む作業をいう。この場合、1つのプログラムユニットまたはデータユニットとは、END文で終了する1連のアセンブラソースプログラムまたはデータである。

実際のパスは2通りの方法で行うことができ、それは、外部ファイル（カセットファイル）を読み込む方法と外部ファイルをシステム内にロードしてそれを読む方法とである。どちらの方法を用いるかはPASS1を行う場合に選択する。外部ファイルを読み込む方法を選択すると、各パスで必ずカセットソースファイルの読み込みを行わなくてはならないが、ソースファイルによってRAMエリアが占有される部分がないので、他の方法より、アセンブル作業の実行できるソースファイルの大きさはずっと大きいことになる。

各パスの作業内容は次の通りである。

- PASS 1 アセンブラソースファイルを読み込み、アSEMBル作業に必要なシンボルテーブルを作成する。パスの方法の選択で"RAM"を指定した場合にはシンボルテーブルの作成とともにアSEMBラソースファイルをそのままRAMエリアにロードし、RAM上にファイルコピーを作成する。
- PASS 2 アSEMBラソースプログラムを読み込み、シンボルテーブルをもとに、アSEMBル作業を行い、アSEMBルリストをテレビ画面上に表示する。アSEMBルリストは、ソースプログラムと、ラインナンバ、相対アドレス、リロケータブルバイナリコード、アSEMBラメッセージとから成る。
- PASS 3 アSEMBルリストのハードコピーをとるために、オプションのプリンタへアSEMBルリストを出力する。リスト上に表示される内容はPASS2のものと同じであるが、オプションプリンタ（放電式またはドット式）は80桁のラインプリンタであり、1命令が1ラインで表示される。（PASS2のテレビ画面上のアSEMBルリストは2行ずつとなる。）
- PASS 4 アSEMBラソースプログラムを読み込み、シンボルテーブルをもとに、アSEMBル作業を行い、リロケータブルバイナリをRAMエリアに作成する。リロケータブルバイナリが作成された後にfilenameで指定する出力ファイルにリロケータブルバイナリファイルを出力する。リロケータブルバイナリファイル上のコードのフォーマットは、命令単位に定義状態を示すフレーム、シンボルフレーム、オペコード、オペランドから構成される。

アSEMBラは以上の作業を機械的に実行するだけであるからエラーメッセージが表示された場合の修正、変更などは、エディタに戻ってソースファイルを編集し直すことになる。

2-2 アセンブラ言語規約

アセンブラソースプログラムは、アセンブラ言語規約に従った文法で記述されなくてはならない。この節では、アセンブラソースプログラムの文構造、文を構成する言語規約について解説を行う。

アセンブラソースプログラムは、そのアセンブル単位であるアセンブラソースプログラムユニットによって構成される。アセンブラソースプログラムユニットとは、END文で終了する1つのソースプログラムであるが、アセンブラは、このプログラムをアセンブル単位としてアセンブル作業を実行して、リロケートブルバイナリファイルユニットを出力することになる。アセンブラソースプログラムが複数のアセンブラソースプログラムユニットから成る時は、アセンブラ出力のそれぞれのリロケートブルバイナリファイルユニットをロードでリンクして構成することになる。

アセンブル単位としての1つのアセンブラソースプログラムユニットは次の各文要素より構成される。

Z80 インストラクションニモニクコード

ラベルシンボル

コメント文（注釈文）

擬似命令	{	定義文
		エントリ文
		リロケート文
		スキップ文（ホーム文）
擬似命令		エンド文

このうちコメント文(注釈文)は、プログラマが適宜置くことのできるものであるが、これはプログラム自体には何ら影響せず、リロケートブルバイナリファイルには情報は残されない。アセンブラソースプログラムユニットは、擬似命令ENDによるエンド文で終わらなくてはならない。

Z80 インストラクションニモニクコードとは、文字通りZ80アセンブリ言語による命令コードのことであり、プログラムの本文を構成するものである。Z80インストラクションニモニクコードは、SYSTEM PROGRAM 別冊"Z80 PROGRAMMING MANUAL"で使用されているコードを用いる。

即ち、最大4桁までのオペコード（CALLとかJPとか）、セパレータ（スペース、コンマなど）およびオペランドの各要素からなる。

ラベルシンボルは、マクロアセンブラで、アドレスまたはデータをシンボリックに表現するためのものであり、ニモニクコードの前のラベル欄に置かれ、セパレータ"："によって命令文から区別される。または命令文のオペランド中に置かれてシンボル参照が行われる。

ラベルシンボルは、最大6文字までが有効である。7文字以上を使用した場合、7文字目以降はシンボルの識別に関係しない。たとえば、ABCDEFGHとABCDEFHは同一のシンボルと見做される。

ラベルシンボルに使用する文字は通常、英数字であるが、セパレータまたは特殊記号として用いられる文字以外の文字も使用することができる。

コメント文は、セパレータ";"以降、[CR]コードまでの文である。プログラム上では注釈としての意味をもつだけである。

擬似命令は、2-4項で詳しく説明される一群のアセンブラ自体に対する命令であり、Z80インストラクションニモニクコードと同様の欄に記述される。擬似命令には、定義文、エントリ宣言文、リロケート文、スキップ文などがある。

エンド文は、擬似命令のうちの1つであるが、アセンブラソースプログラムユニットはこのEND文によって終わらなくてはならない。

文字, 記号 (character)

アセンブラソースプログラムで使用される文字は、英数字、特殊記号、その他のキャラクタであり、このうち特殊記号はアセンブラに機能的に働きかけるものである（セパレータ":", ";"や **CR** , **SPACE** コードなどがそれである）。

前頁に示した、アセンブラソースプログラムの各文要素は、これらの文字、記号から構成される。

1) 英文字 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

シンボルや、インストラクションモニタコードを表現するために用いられる。また、A～Fの6文字は16進の数値としても使用される。またDはdecimal (10進)、Hはhexa (16進)の意味にも使われる。

2) 数 字: 0 1 2 3 4 5 6 7 8 9

定数 (10進数または16進数) や、シンボルを表現するために用いられる。定数が、10進表現であるか16進表現であるかは、後に示す定数規約の約束に従って区別される。

3) スペース (空白, SPACE コード)

テキストエディタのTコマンドでソースリストを表示すると、スペースはそのまま表示されるが、アセンブラでは、コメント文中以外ではセパレータとなり、アセンブルリスト表示でタブレーション機能を持つ。スペースコードがセパレータとなるのは、オペコードとオペランドの間、およびオペランドとコメント欄の間（或いは、オペコードとコメント欄の間）に置かれた場合である。

(例)

	OR	[SP]	F0H	[SP]	;A←X0	} エディタリスト
	XYZ :	PUSH		[SP]	AF	
	ADD	[SP]	HL, BC	[SP]	;BC=COUNT	

↓

	OR		F0H		;A←X0	} アセンブラリスト
XYZ :	PUSH		AF			
	ADD		HL, BC		;BC=COUNT	

↑
タブセット

↑
タブセット

4) コロシ " ; "

ラベルシンボルと、命令文（オペコードまたは擬似命令）の間に置かれた場合セパレータとなる。アセンブルリスト表示でタブレーション機能をもつ。

```
(例)   START: LD    SP, START
        MAIN:  ENT
        ↑      ↑
        タブセット タブセット
```

この場合、命令文が無くてもラベルシンボルとしてアドレスを定義させることができる。(シンボルの項を参照)。

(例) ENTRY: ← "ENTRY" は "TOP0" と同一のアドレスに定義される。
TOP0: PUSH HL

5) セミコロンの使用法

コメント文（注釈文）の始まりを表明するものであり、この後、行の終りまではプログラムには全く関係しない。行の先頭または、コメント欄の先頭に用いる。

```
(例)      ;  
          ; SAMPLE PROGRAM  
          ;  
          CMMNT: ENT          ; COMMENT  
                                コメント欄
```

行の全体がコメント

6) キャリッジリターン (CR コード)

行の終りを示す。

7) その他の特殊記号 \backslash () ,

\backslash () , などは特殊記号として、命令文中で使用される。

8) その他のキャラクタ

システムに用意されている、その他のキャラクタ、たとえばグラフィックパターンやカナ文字などは、コメント欄、ラベルシンボルなどに使用することもできるが、一般には使用しない。

カーソルコントロール用のキャラクタを、ソースプログラム中で記述すると、テレビ画面のリスト表示の際に、カーソルコントロールが実行されてしまうこともある。(たとえば、"C"コードを使っていると、そのリスト表示の際、画面がクリアされ、カーソルホームが実行される。それを避けるために、キャラクタ "C" を用いずに、アスキーコード16Hを用いることができる)

行 (line)

アセンブラソースプログラムの1行は、英数字、記号などで構成され、キャリッジリターンで区切られる。1行中に含まれる文は、コメント文を除くと、Z80インストラクション命令文、擬似命令文、エンド文、もしくはスキップによる空白、のいずれか1つの文である。

1行は、アセンブルリスト表示の際に、その構成要素にしたがったタブセットによって配置される。(P.14のアセンブルリスト表示を参照)

シンボル (symbol)

シンボルは、特殊記号以外の文字、記号を用いて作ることができるが、一般には英数字を用いる。シンボルは、6文字以内で構成されなくてはならない。7文字以上のシンボルを用いても、7文字目以降はアセンブラでは無視される。

(例)	よい例	ABC	START	BUFFER	50STEP	
	悪い例	(ABC)	,HL	IY+3	XYZ+3	←特殊記号を用いている。
	7文字以上	COMPARE0	←	同一のシンボル"COMPAR"として取り扱われる。		
		COMPARE1	←			

シンボルのデータ定義(1バイトまたは2バイトデータ)は擬似命令EQUによって行われる。

(例)	ABC:	EQU	3
	CR:	EQU	0DH
	VRAM0:	EQU	D000H

シンボルをリロケータブルアドレスに定義するには、命令文の前に、セパレータ": "を置いて定義すればよい。グローバルシンボルとして定義するには、擬似命令ENTを用いる。

(例)	RLDR:	ENT
	RLDR0:	PUSH HL

シンボルをオペランドで用いる場合、たとえば、CALL ABCという命令の場合、アセンブラは、オペランド"ABC"をはじめシンボルとして、シンボルテーブルを検索する。シンボルテーブルに"ABC"が見つからなかった場合は、16進データ"0ABCH"と見做して、0ABC番地をコールすることになる。

シンボルを参照する場合(即ち、オペランドで使用する場合)、そのシンボルは同一のアセンブラソースプログラムユニットで定義されているか、外部ユニット中でグローバル定義がなされていなければならない。そうでない場合、シンボル値は、未定義のままバイナリコード化される。

一度定義されたシンボルを再度定義することはできない。

シンボルのリロケートブルアドレス定義には次のような使い方もできる

(例)

```
ABCD:  ENT
EFGH:  ENT
IJK:   LD   A, B
```

シンボル "ABCD" "EFGH" "IJK" はいずれも、LD A, B のオペコードアドレスに定義される。このうち "ABCD" と "EFGH" はグローバルシンボルである。

```
ABCD:
EFGH:
IJK:   LD   A, B
```

上と異なる点は、"ABCD" "EFGH" はグローバルシンボルではない点である。

(注) 外部宣言 (ENT, EQU) に使われるシンボルについては、外部からオフセットをつけた参照を行うことはできない。

(例)

```
XYZ: EQU 6000H
ABC: ENT
      LD   A, B
      LD   HL, XYZ + 3  ← 不可
      JP   DEF + 3      ← 不可
DEF: ENT
      JP   ABC + 1      ← 不可
      LD   DE, XYZ + 10 ← 不可
      JP   ABC
      END
```

定数 (constant)

定数には、10進数と16進数とがある。符号は、" + " " - " を用いることができるが、符号の無い場合は " + " の省略形と見做す。

アセンブラは定数 (キャラクタは 0 1 2 3 4 5 6 7 8 9 A B C D E F + - D H から構成される) を読み、数字だけで構成されている時は、10進と見做す。また最後の桁が " D " で他の桁がすべて数字の場合も10進と見做す。

(例) 次の定数はいずれも10進数である。

```
23 999 +3 -62 16D 0003D
      16      3
```

それ以外の場合は16進数と見做すが、原則として16進数の場合は、最後に " H " をつけなくてはならない。

(例) 次の定数はいずれも16進数である。

```
2AH CDH +01H -BH 0010H 00ADH 00H
```

2-3 アセンブルリストとアセンブラメッセージ

アセンブラのPASS2、PASS3を実行すると、それぞれテレビ画面上、オプションプリンタ上へアセンブルリストがタイプアウトされる。アセンブルリストを検討することは、アセンブラプログラミングで最も重要な過程の1つであると言える。即ち、作成したアセンブラソースプログラム中に誤りがないかどうか、あるいは、所望の機械語コードが得られているか等の基本的なチェックを行うものだからである。

アセンブラは、アセンブラソースプログラムユニットをアセンブルして、アセンブルリスト上へ、ラインナンバ、相対アドレス、リロケートブルバイナリコード、アセンブラメッセージ、アセンブラソースプログラム（ラベルシンボル、Z80インストラクションニモニクコード、コメント文）を表示する。またアセンブルリストは、ページングが行われ、1ページに50行表示されると改ページされる。各行は、一般に80桁／行の形式で表示が行われ、テレビ画面上では1行が、実際には2行にわたって表示されるが、オプションプリンタは80桁のラインプリンタであるから1行1ラインに表示される。

アセンブルリストの構成を次に示す。

ライン ナンバ	相対アドレス ↓ リロケートブル バイナリコード	アセンブラメッセージ ↓ ラベル欄	オペ コード	オペランド	コメント欄
** Z80 ASSEMBLER SP-2101 PAGE 01 ** <input type="checkbox"/> ページングの際この表示がなされる。					
01	0000	:	:	:	:
02	0000	:	:	:	:
03	0000	:	:	:	:
04	0000	:	:	:	:
05	2000	P	LETNL:	EQU 0006H	: RELOCATION
06	2000	P	MSG:	EQU 0015H	:
07	2000	:	:	:	:
08	2000	:	:	:	:
09	2000	:	:	:	:
10	2000	310020	LD	SP,START	: ENTRY FROM UNIT#1
11	2003	210000	E	LD HL,TEMPO	: ENTRY FROM UNIT#2
12	2006	DD210000	E	LD IX,TEMP1	: INITIAL STACK POINTER
13	200A	DD360000	EE MAINO:	LD (IX+CONSTO),CONST1	: CONST1=F
14	200E	00	Q	XOA A	: A<--00
:	:	:	:	:	:
47	205A	1A	MAIN7:	LD A,(DE)	:
48	205B	B7		OR A	:
49	205C	2000	V	JR NZ,COMP	: EXCHANG DE,HL
50	205E	EB	MAIN8:	EX DE,HL	:
** Z80 ASSEMBLER SP-2101 PAGE 02 ** <input type="checkbox"/> 前ページが50行になると改ページされる。					

ラベル欄、オペコード欄、オペランド欄、コメント欄のそれぞれにタブセットが行われていることがわかる。テレビ画面へのリスト表示では、ラインナンバからアセンブラメッセージ欄までが第1行目に表示され、ラベル欄からコメント欄までが第2行目に表示される。（オペランドまたはコメント欄の長さによって3行以上になることもある。）

従って、例えば上のリストのラインナンバ13の行は、テレビ画面上では次のように表示される。

```
13 200A DD360000 EE
    MAINO: LD (IX+CONSTO),CONST1 ; CONST1=F
```

アセンブルリスト上のアセンブラメッセージ欄に表示されるメッセージは、アセンブル作業を実行した際に検出された、アセンブルソースプログラムユニット中の誤り、あるいは定義状態を指摘しておくべき点について行われる。プログラマは、これらのアセンブラメッセージによって、リロケータブルバイナリに登録される時の定義状態、ソースプログラム中のエラー等の情報を得る。

——定義状態を示すメッセージ——

E (External)

エクスターナルシンボル参照が行われていることを示すメッセージである。即ち、オペランドで参照しているラベルシンボルがそのアセンブラソースプログラムユニット中に定義されていないことを示す。

従って、E表示のなされたラベルシンボルは、外部アセンブラソースプログラムユニット中で、グローバルシンボルとしてのエントリ宣言（擬似命令 ENT P.17 参照）がなされていなければならない。外部プログラムユニットとロードでリンクされて、はじめてラベルシンボルの参照が成立することになる。

エクスターナルシンボル参照が成立せず、未定義のままバイナリプログラムを得た場合、未定義の1バイトデータは"00"、2バイトデータ（またはアドレス）は"FFFF"となる。

```
(例)      E   LD    B, CONST0
           ↑
           1 バイトデータ "CONST0" が未定義である。
E   CALL  SORT
           ↑
           アドレス "SORT" が未定義である。
EE  BIT   TOP, (IY+FLAG)
           ↑
           1 バイトデータ "FLAG" が未定義である。
           |
           1 バイトデータ "TOP" が未定義である。
```

P (Phase)

擬似命令 EQU (P.19参照) によって定数が定義されたラベルシンボルを表わす。P表示のなされたラベルシンボルは、外部ファイルによって参照することもできるが、そのリンキングには条件が付随する。(P.61参照)

P表示は、PASS 1 と異ったラベル値が検出された場合も行われる。

```
(例)      P      LETNL: EQU 0006H
P      DATA1: EQU 3
           ↑
           "LETNL", "DATA1" が EQU によって定義されていることを示す。
P表示は、アセンブラメッセージ欄ではなく、リロケータブルバイナリコードの欄に表示される。
```

——エラーメッセージ——

C (illegal Character error)

オペランドに、アセンブラ規格外のキャラクタが記述されたことを示す。

```
(例)      C   JP  +1000-3
```

F (Format error)

命令のフォーマットが異なることを示す。

N (Non label error)

擬似命令 ENT, EQU でラベルシンボルがないことを示す。

```
(例)      N      EQU 0012H
           _____
           ラベルシンボルがない。
```

L (erroneous Label error)

アセンブラ規格外のラベルシンボルが記述されたことを示す。

(例) L JR XYZ

↑ "XYZ" は該当アセンブラソースプログラムユニット中にはない。

JR, DJNZ コマンドでは外部シンボルの参照を行うことはできない。該当プログラム中にな
いラベルが参照されると、外部シンボル参照と見做して" L " 表示を行う。

L CONT 1 : EQU CONT 2

EQU 命令ではオペランドにラベルを用いることはできない。

M (Multiple label error)

該当アセンブラソースプログラムユニット中で、同じラベルシンボルを2回以上定義した。

(例) M ABC: LD DE, BUFR

3

M ABC : ENT

[↑] "ABC" が重ねて定義されていることを示す。

O (erroneous Operand)

アセンブラ規格外のオペランドが記述されたことを示す。

Q (Questionable mnemonic)

命令のパターンが合わないことを示す。

(例) Q CAL XYZ

CALL XYZ としなくてはならない。

Q PSH B

PUSH BC としなくてはならない。

S (String error)

擬似命令 DEFM で, " ' " が足りない。

(例) S DEFM GAME OVER

DEFM 'GAME OVER' としなくてはならない。

V (Value over)

オペランドに、命令規格を越えた値が記述されたことを示す。

(例) V LD A, FF8H

V SET 8, A

V JR -130

2 — 4 アセンブラ擬似命令の考え方

擬似命令は、Z80命令セットとは異なり、アセンブラに対する命令である。アセンブル作業に指示を与えるだけで、命令そのものがZ80機械語に変換されることはない。ただし、DEFB、DEFW、DEFM命令では、そのオペランドが機械語に変換されることはある。その他の擬似命令は、ラベルシンボルを有効に使ったりアセンブルリストの様式を決めるなどのために用意されている。

ENT (entry)

ラベルシンボルの、グローバルシンボルとしてのエントリ宣言命令である。即ち、幾つかのプログラムをリンクする場合、相互に参照し合うラベルシンボルは、この宣言がなされていなくてはならない。

エントリ宣言のなされたラベルシンボルは、アセンブラ出力のリロケートブルバイナリファイル中に、情報として残り、ローダでのリンク操作を可能にするのである。また、シンボリックデバッガでは、このラベルシンボルを用いた、シンボリックなアドレス指定が可能となる。

エントリ宣言のなされていないラベルシンボルは、その1つのプログラム内でのアセンブル作業に寄与するだけであり、リロケートブルバイナリファイルへ、情報が残されることはない。ただし、擬似命令EQUは、ラベルシンボルのグローバル宣言を含んでいるので、エントリ宣言は不要である。

下図は2つのプログラムユニット "GAUSS-MAIN" と "GAUSS-SR" 間でのラベルシンボルの参照のようを示している。

アセンブルリスト上に示される "E" 表示は、該当プログラム中にないラベルシンボルが参照されていることを示すものであり、"external" の意味を持つ記号である。

プログラムユニット 1
"GAUSS-MAIN"

```
                ; GAUSS-MAIN
                ;
                MAIN0 : ENT                ←ラベルシンボル
                                           "MAIN0" の
                                           エントリ宣言
                :
                CD0000 E                  CALL CMPLX
                :
                                           external表示
                :
                END                        ←プログラムユニットの
                                           最後はEND文が必要
```

プログラムユニット 2
"GAUSS-SR"

```
                ; GAUSS-SR
                ;
                CMPLX : ENT                ←ラベルシンボル
                                           "CMPLX" の
                                           エントリ宣言
                :
                RET
                :
                C30000 E                  JP    MAIN0
                :
                                           external表示
                :
                END
```

REL nn' (relocate)

アセンブラ自身が持っているリファレンスカウンタの内容に、nn'を加算する命令である。リファレンスカウンタとは、アセンブル作業に必要な、アドレスを割り振るためのカウンタであり、通常、0000から開始される。従って、通常のアセンブルリスト上に示されるアドレスは、冒頭が0000番地となって、順次オペコードのバイト数ずつカウントアップされて行く。また、アセンブラ出力のリロケートブルバイナリファイルの内容もその形式となっている。それに対して、プログラムの冒頭に、たとえば

```
REL 1200H
```

といった、REL命令をおくと、アセンブル形式は、RELで指定された1200番地 (Hex) を、アセンブルバイアスとして行なわれる。目的プログラムを、最終的に1200番地から構成したい場合、目的プログラムの機械語と同一のコードを、このようにしてアセンブルリスト上に見ることができ、便利である。

注意が必要なのは、このようにREL命令を行なっているリロケートブルバイナリファイルをリロケートブルローダに入力する場合である。

リロケートブルローダで指定するアセンブルバイアスは、リロケートブルバイナリファイル中の相対アドレスに加算されるものであるから、既に、REL命令を用いて、目的とするアドレス形式のリロケートブルバイナリをアセンブラで作成している場合、ローダ側で与えるアセンブルバイアスは0000としなくてはならない。

REL命令をプログラムの冒頭でなく、中間で使用した場合は、そこでリファレンスカウンタのカウントアップが行なわれるが、擬似命令DEFSの場合と異なり、アドレスのカウントアップ（アドレスの飛び）は行なわれない。ただし、アセンブルリスト上に見られるアドレスはカウントアップされている。これは、アセンブルリスト上のアドレスとは、リファレンスカウンタの内容であり、見かけ上のアドレスに過ぎないからである。

*** Z80 ASSEMBLER SP-2101 PAGE 01 ***

```
01 0000      :
02 0000      :  EXPERIMENT OF REL NN
03 0000      :
04 0000      REL    1200H
05 1200      START: ENT
06 1200 310012      LD    SP, START    ;  INITIAL SP
07 1203 CD4700      CALL  MSTP         ;  MUSIC STOP (0047H)
```

——
アドレス表示は
RELのあとから
1200Hとなっ
ている。

EQU (equate)

ラベルシンボルを、数値データ（アドレスの場合もある）に定義する命令である。この場合、数値データとは絶対的な定数であり、10進数または16進数でなくてはならない。

通常のアセンブル過程では、オペランドにアドレスとしてラベルシンボルが用いられると、相対アドレスとして取り扱われるが、そのラベルシンボルが、EQUによって具体的に定まったアドレスに定義されていたら、アセンブル過程でその値が変化することはない。

EQU命令は、ラベルシンボルのグローバル宣言を含んでおり、ENT宣言を使用しなくても外部プログラムからの参照が可能である。しかし、この場合、EQU命令を含むプログラムユニットは他からリンクするプログラムユニットより先にロードする必要がある。

これらの特徴から、たとえば次のような使い方に適切である。すなわち、次に示すようにモニタサブルーチンのアドレスや、特定の出力デバイスのI/Oポート番号などを、適当なラベルシンボルに定義するといった使い方である。但しここでアセンブラメッセージ"P"は、EQU命令によるラベルシンボルの定義がなされていることを示す。(phase)

** Z80 ASSEMBLER SP 2101 PAGE 01 **

```

01 0000      ;
02 0000      ; MONITOR SUBROUTINE
03 0000      ;
04 0000 P    PRNT:      EQU  0012H
05 0000 P    PRNTS:     EQU  000CH
06 0000 P    NL:        EQU  0009H  ; 次の LETNL と異なるのはテレ
07 0000 P    LETNL:     EQU  0006H  ビ画面の行の先頭にカーソルが
08 0000 P    MSG:       EQU  0015H  あると行替えは実行されない点
09 0000 P    GETL:      EQU  0003H  である。
10 0000 P    GETKY:     EQU  001BH
11 0000 P    BRKEY:     EQU  001EH
12 0000 P    MELDY:     EQU  0030H
13 0000 P    BELL:      EQU  003EH
14 0000 P    XTEMP:     EQU  0041H
15 0000 P    MSTA:      EQU  0044H
16 0000 P    MSTP:      EQU  0047H
17 0000 P    TIMST:     EQU  0033H
18 0000 P    TIMRD:     EQU  003BH
19 0000      SKP  3

23 0000      ;
24 0000      ; SET PORT# : PRINTER
25 0000      ;
26 0000 P    POTFE:     EQU  FEH
27 0000 P    POTFF:     EQU  FFH

```

DEFB n (define byte)

この行の位置(アドレス)に、バイト定数 **n** をそのままセットする。 **n** のかわりにバイト定数の定義されたラベルシンボルを使うこともできる。

この命令およびDEFW, DEFMなどは特に、メッセージデータ群やグラフィックデータ群を構成したり、コード変換用のテーブルやその他のデータテーブルの構成などに使用されることが多い。

次に示す例は、DEFBを用いて、メッセージ "ERROR" をアスキーコードで構成したものである。エンドマークとして0Dコードを置いているので、モニタサブルーチン 0015H を用いてメッセージアウトが可能となる。

13	1FF3	B7		OR	A
14	1FF4	CA0000	E	JP	Z, READY
15	1FF7	110020		LD	DE, MESGO
16	1FFA	CD0000	E	CALL	MSG
17	1FFD	C30000	E	JP	MAIN2
18	2000		;		
19	2000		;	MESSAGE GROUP	
20	2000		;		
21	2000		MESGO: ENT		; "ERROR"
22	2000	45		DEFB	45H
23	2001	52		DEFB	52H
24	2002	52		DEFB	52H
25	2003	4F		DEFB	4FH
26	2004	52		DEFB	52H
27	2005	0D		DEFB	0DH

DEFB 'S' (define byte)

この行の位置(アドレス)に、1文字 " S " のアスキーコードをセットする。
この命令では、文字をアスキーコードに変換するので、前例のMESGOは次のように構成することもできる。

21	2000		MESGO: ENT		; "ERROR"
22	2000	45		DEFB	' E '
23	2001	52		DEFB	' R '
24	2002	52		DEFB	' R '
25	2003	4F		DEFB	' O '
26	2004	52		DEFB	' R '
27	2005	0D		DEFB	0DH

DEFW nn' (define word)

この行の位置(アドレス)にn', 次の位置にnをセットする。即ち、2 バイトのデータをセットする命令である。
nn' のかわりに、ラベルシンボルを用いることもできる。

```
39 5FF1          CMDT:  ENT          ;  COMMAND  TABLE
40 5FF1  41          DEFB  41H
41 5FF2  0053        DEFW  CMDA
42 5FF4  42          DEFB  42H
43 5FF5  1E53        DEFW  CMDB
44 5FF7  53          DEFB  53H
45 5FF8  0000  E      DEFW  CMDS
46 5FFA  0D          DEFB  0DH
47 5FFB          CONST0: ENT
48 5FFB  0F01        DEFW  010FH
49 5FFD          CONST1: ENT
50 5FFD  660D        DEFW  0D66H
```

DEFM 'S' (define message)

この行の位置(アドレス)から、文字列S (ストリング)をアスキーコードで順次セットする。文字列Sの文字数は1 から64までの範囲である。アセンブルリスト上には4 文字分ずつコード化された形で表示される。

この命令を用いると、前例で用いたメッセージ"ERROR"は次のようにして作成することができる。

```
21 2000          MESGO:  ENT          ;  "ERROR"
22 2000  4552524F    DEFM  'ERROR'
23 2004  52
24 2005  0D          DEFB  0DH
```

DEFS nn' (define storage)

この行の位置(アドレス)から、nn'バイト分だけメモリ領域を確保する。

この命令では、リファレンスカウンタにnn'バイト加算し、同時に該当するアドレスの範囲内を、未定義のまま飛ばす。

(RELの場合は、単にリファレンスカウンタにnn'バイトを加算するだけで、メモリ領域の確保は行なわれない。)

たとえば、バッファとして用いるエリアを構成する場合に次のような使用法が考えられる。

```
02 4BB8          TEMPO:  ENT          ; BUFFER A
03 4BB8          DEFS    1
04 4BB9          TEMP1:  ENT          ; BUFFER B
05 4BB9          DEFS    2
06 4BBB          TEMP2:  ENT          ; BUFFER C
07 4BBB          DEFS    2
08 4BBD          TEMP3:  ENT          ; BUFFER D
09 4BBD          DEFS    128
10 4C3D          BFFR:   ENT          ; BUFFER E
11 4C3D          DEFS    A
12 4C47          BUFFER: ENT          ; BUFFER F
13 4C47          DEFS    2
    _____
```

アドレスがDEFSで指定された値 (10進数または16進数) だけカウントアップされているのがわかる。

SKP n (skip n lines)

アセンブルリストを出力する場合に、n 行ぶんだけ行送り (line feed) をする。これによってリスト表示を適当に区切り、見易くすることができる。

```
30                                COMMON: ENT                        ; NORMAL RETURN
31 3BB8 AF                        XOR A                            ; A<--- 00
32 3BB9 32B84B                    LD (TEMPO), A `; CLEAR CMD BUFFER
33 3BBC 110020                    LD DE, MESGO ; "READY"
34 3BBF C9                        RET
35 3BC0                            SKP 3

                                     } 3行分の行送り
                                     } が行なわれる

39 3BC0                            :
40 3BC0                            ; ABNORMAL RETURN
41 3BC0                            :
42 3BC0                            ABNRET: ENT                        ; SET INVALID MODE
```

SKP H (skip home)

アセンブルリストを出力する場合に、ページ送りをする。

END (end)

1つのソースプログラムユニットの終端を宣言する。ソースプログラムユニットは必ずこのEND文で終了しなければならない。END文が無いと、アセンブル作業が完了されないことになる。

アセンブラのPASS 1で、END文の無いソースファイルを読み出した場合

END ?

の表示がなされる。

2 — 5 アセンブラの各PASS

PASS 1

ソースファイル(テキストエディタ出力のカセットファイル)を読み込み、シンボルテーブルを作成する。PASS 2からPASS 4までのアセンブル作業は、2通りの方法で行うことができる。すなわち、ソースプログラムの読み込みをカセットファイルに対して行うか、RAMエリアにローディングされたソースプログラムに対して行うかの2通りであり、PASS 1で、どちらの方法を用いるか選択する。後者を選ぶと、PASS 1の実行時に、シンボルテーブルの作成と並行して、ソースファイルのRAMエリアへのローディングも行われる。

PASS 1

CASSETTE (1), RAM (2) ?1

FILENAME? GAUSS MAIN **CR**

↓ PLAY

FOUND GAUSS MAIN

LOADING GAUSS MAIN

ソースファイル "GAUSS MAIN" の読み込みを行い、アセンブル作業に必要なシンボルテーブルを作成せよ
アセンブル作業は、CASSETTE、すなわち各 PASS で読み込むソースプログラムを、カセットファイルとして行う

PASS 1

CASSETTE (1), RAM (2) ?2

FILENAME? GAUSS S-R **CR**

↓ PLAY

FOUND GAUSS S-R

LOADING GAUSS S-R

ソースファイル "GAUSS S-R" の読み込みを行い、アセンブル作業に必要なシンボルテーブルを作成せよ
アセンブル作業は、RAM、すなわち各 PASS で読み込むソースプログラムを、RAMエリアにローディングされたソースプログラムとして行う
従って、ソースファイル "GAUSS S-R" のRAMエリアへのローディングも実行せよ

—— "PASS" のコマンド待ちに、1を入力する。ソースプログラムのアセンブル作業は、必ずPASS 1によって開始されなければならない。

PASS 1を行う時、システムは一担RAMエリアをクリアし前のアセンブル作業で使用したシンボルテーブルなどをシステムから取り除いた上でソースファイルの読み込みを行うことになる。

—— システムは、"CASSETTE (1), RAM (2)" の表示を行い、アセンブル作業を、どちらの方法で行うかを訊く。CASSETTEを用いて、各パスでソースファイルの読み込みを行う場合には"1"を指定する。あるいはRAMを用いて、PASS 1でカセットファイルのRAMエリアへのローディングを実行してアセンブル作業を行うには"2"を指定する。

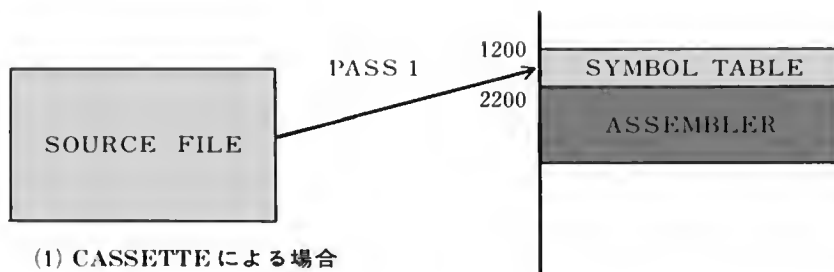
ソースプログラムが、システムのRAMエリアの大きさに対して、そう大きくない場合は、通常アセンブル作業はRAMを用いて行う。すなわち、ソースプログラムをロードし、PASS 4でリロケータブルバイナリをメモリ内に構成するのに十分なRAMエリアがある時は、そのエリアにソースプログラムをロードしておいて、他のPASSでは、いちいちカセットファイルの読み込みを行わなくて済むからである。

ソースプログラムが大きくて"RAM"の方法が使えない場合、CASSETTEの方法をとる。この場合システムは、他の各PASSに於いて、カセットファイルを一定バイトずつ読み込み、アセンブル作業を進めること

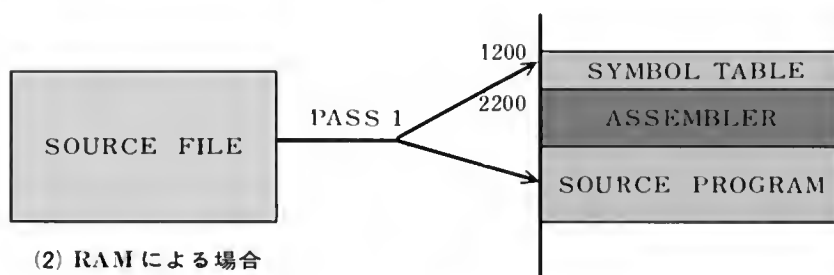
になる。一般に、ソースファイル中のデータのバイト数とリロケートブルバイナリファイル中のデータのバイト数とでは、リロケートブルバイナリの方がずっとバイト数が小さいので、2通りの方法でそれぞれアセンブルできるファイルの大きさはかなりのひらきがある。

- アセンブル作業の方法を指定すると、システムは "FILENAME?" と表示して読み込むソースファイルのファイル名の指定を待つ。
 - ファイル名を指定したら **[CR]** キーを押す。
 - [CR]** キーが押されたら、PLAYボタンを押すよう指示される。
 - PLAYボタンを押すと、ファイル名で指定されたソースファイルを見つけ出し、読み込みを始める。ファイル名が指定されていない場合は、最初に見つけ出したソースファイルの読み込みを行う。
 - アセンブル作業は、各PASSについて、END文によって終了する。従って、各ソースプログラムユニットは終りにEND文が置かれていなくてはならない。
- END文の無いソースファイルを読み込んだ場合、"END?" の表示がなされる。
- ソースファイルの読み込みを中止するには **[SHIFT] [BREAK]** を押す。
 - PASS 1 を終了すると、次のPASSの指定を待つ。

- PASS I の実行形態を下図に示す。ソースファイルを読み込んで作成するシンボルテーブルの容量は約4 K バイト弱であり、メモリマップ上で、アセンブラの前 (1200から2100) に置かれている。従って、ソースプログラムで使用できるシンボルの総数には限度があることになる。



(1) CASSETTE による場合



(2) RAM による場合

- 右の写真は、アセンブラをロードし、PASS I で、CASSETTEの方法を指定してソースファイル "GAUSS MAIN" の読み込みを行うもようを示している。

```
** MONITOR SP-1002 **
*LOAD
↓PLAY
LOADING 280 ASSEMBLER
PASS 1
CASSETTE(1),RAM(2) ?1
FILENAME?GAUSS MAIN
↓PLAY
FOUND GAUSS MAIN
LOADING GAUSS MAIN
PASS
```

PASS 2

ソースプログラムをアセンブルして、アセンブルリストをテレビ画面上に表示する。アセンブルリストは、2—3項で述べたように、ページ、ラインナンバ、相対アドレス、16進機械語、アセンブラメッセージ、アセンブラソースプログラム(ラベル、ニモニック、データ、注釈)の順に各欄に表示されるが、テレビ画面上ではそれぞれ2行にわたって表示が行われる。アセンブリ語ソースプログラムと機械語との対照、アセンブラメッセージの参照など、アセンブラプログラミングで最も重要なチェック過程の一つである。

PASS 2

FILENAME? GAUSS MAIN **[CR]**

↓ PLAY

アセンブル作業を実行して、アセンブルリストをテレビ画面へ出力せよ

読み込みを行うカセットソースファイルは " GAUSS
MAIN " である
(CASSETTEの場合)

PASS 2

RAMエリアのアセンブラソースプログラムをアセンブルして、アセンブルリストをテレビ画面に出力せよ
(RAMの場合)

— " PASS " のコマンド待ちに、2を入力する。

PASS 1で、アセンブル作業の方法をCASSETTEに選んだ場合、システムは、ソースファイルの読み込みのために、読み込むべきファイルネームの指定を待つ。

ファイルネームを指定し、**[CR]**キーを押すと、PLAYボタンを押すよう指示される。

PLAYボタンを押すと、システムは読み込むべきソースファイルを見つけ出してソースプログラムを読み込み、アセンブル作業を始める。ファイルネームを指定していないと、最初に見つけ出したソースファイルの読み込みを始める。

ソースファイルは256バイトずつ読み込まれてアセンブル作業が行われ、テレビ画面上へアセンブルリストの表示が行われる。(カセットレコーダは動いたり止ったりすることになる)

テレビ画面へのリスト表示は、**[SPACE]**キーにより中断、続行が可能であるから、任意の位置のリスト表示を詳しく検討することができる。

PASS 1で、アセンブル作業の方法をRAMに選んだ場合は、即座にアセンブルリストの表示が開始される。リスト表示の中断、続行は、**[SPACE]**キーによる。

PASS 2を中止する場合 **[SHIFT] [BREAK]**を押す。

— 右の写真は、PASS 1でCASSETTEによる方法を指定し、ソースファイル " GAUSS MAIN " のシンボルテーブルを作成した後に、PASS 2を実行する時のもようを示す。

RAMの方法では、PASS 2を入力すると直ちに、アセンブルリストの表示が開始される。

```
PASS 1
CASSETTE(1),RAM(2) ?1
FILENAME?GAUSS MAIN
↓ PLAY
FOUND GAUSS MAIN
LOADING GAUSS MAIN

PASS 2
FILENAME?GAUSS MAIN
↓ PLAY
```

PASS 3

ソースプログラムをアセンブルして、アセンブルリストをオプションのプリンタ上へタイプアウトする。アセンブルリストは、2 - 3項で述べたように、ページ、ラインナンバ、相対アドレス、16進機械語、アセンブラメッセージ、アセンブラソースプログラムの順に各欄に表示される。その場合、オプションプリンタは80桁のプリンタであるから、テレビ画面への2行にわたる表示も1行で行われる。プリンタへの出力は、アセンブルリストのハードコピーを得るためのものであるから、プログラムの詳細な検討はテレビ画面上のリストを用いるよりも容易となる。

PASS 3

FILENAME?GAUSS MAIN **CR**

↓ PLAY

アセンブル作業を実行して、アセンブルリストをオプションプリンタへ出力せよ
読み込みを行うカセットソースファイルは "GAUSS MAIN" である
(CASSETTE の場合)

PASS 3

RAM エリアのアセンブラソースプログラムをアセンブルして、アセンブルリストをオプションプリンタへ出力せよ
(RAM の場合)

—— "PASS" のコマンド待ちに、3を入力する。

—— PASS 1 でアセンブル作業の方法を CASSETTE に選んだ場合、システムはソースファイルの読み込みのために、読み込むべきファイルネームの指定を待つ。

ファイルネームを指定し、**CR** キーを押すと、PLAY ボタンを押すよう指示される。

PLAY ボタンを押すと、システムは読み込むべきソースファイルを見つけ出してソースプログラムを読み込み、アセンブル作業を始める。ファイルネームを指定していないと、最初に見つけ出したソースファイルの読み込みを始める。

—— ソースファイルの読み込みは256バイトずつ行われる。

—— PASS 1 でアセンブル作業の方法を RAM に選んだ場合は、即座にアセンブルリストのプリンタ上へのタイプアウトが開始される。

—— PASS 3 ではプリンタのトラブルによるメッセージとして次のものがある。

NO POWER OR NO CONNECTION (PRINTER)

……………プリンタが OFF あるいは、接続されていない場合。

ALARM (PRINTER)

……………メカ的なトラブルが発生した場合。

PAPER EMPTY (PRINTER) ……………プリンタ用紙ぎれの状態。

—— プリンタへのリスティングを中止するには、**SHIFT** **BREAK** キーを押す。

——オプションの放電プリンタ（MZ-80 P 2）によるアセンブルリスト出力例を次に示す。このサンプルは、16行のソースプログラムになっている。PAGE 01 にこのプログラムのアセンブルリストがタイプアウトされ、PAGE 02 に、シンボルテーブルが表示される。（1 ページは50行であるから、下図は、各ページの一部分を示しているに過ぎない）

実際の動作では、プリンタは更にもう 1 ページ分ページ送りを実行して止まり、システムは " PASS " のコマンド待ちに戻る。

Z80 ASSEMBLER SP-2101 PAGE 01

```

01 0000      ;
02 0000      ; ASSEMBLER LIST SAMPLE
03 0000      ;
04 0000      ;
05 1200      REL      1200H
06 1200      START:  ENT
07 1200      MAIN1:  ENT
08 1200      LD      SP,START      ;INITIAL STACK POINTER
09 1200      CD4700      CALL  MSTP      ;MUSIC STOP
10 1200      3E05      LD      A,5
11 1200      CD0000      CALL  XTEMP      ;SET TEMPO TO 5
12 1200      CD0000      CALL  CLTBL      ;CLEAR TABLE
13 1200      00      XOA      A
14 1200      321212      LD      (?TABP),A      ;INITIAL I/O #1
15 1212      P      EQU      0047H      ;MUSIC STOP (MONITOR)
16 1213      ?TABP:  DEFS      1
                  END

```

Z80 ASSEMBLER SP-2101 PAGE 02

?TABP 1212 MAIN1 1200 MSTP 0047 START 1200

■ 上に示したアセンブルリストについて次のことを検討せよ

- 1) アセンブルリストの各欄は、それぞれ何桁となっているか。
- 2) 相対アドレスは、何番地から開始されているか。
- 3) REL コマンドが置かれていると、相対アドレスはどのように変わるか。
- 4) " E " 表示のある外部シンボルを参照する命令では、機械語オペランドはどうなっているか。
- 5) " Q " 表示のある命令では、オペコード欄はどうなっているか。
- 6) EQU によるシンボル定義でのメッセージ " P " は、どの位置になされているか。また、そこで定義されたシンボルは、確かに絶対アドレスとなっているか調べよ。(CALL MSTP 命令の機械語オペランド、および、シンボルテーブルを調べよ。)

PASS 4

ソースプログラムをアセンブルして、カセットファイルへリロケータブルバイナリを出力する。リロケータブルバイナリの内容は、アセンブルリストにタイプアウトされた機械語コードおよび、その命令の定義状態を示すインフォメーション、グローバルシンボル等のデータベースによって構成されている。

```
PASS 4
FILENAME?GAUSS MAIN [CR]
↓ PLAY
FOUND GAUSS MAIN
LOADING GAUSS MAIN
OK
FILENAME?GAUSS MAIN/RB [CR]
↓ RECORD.PLAY
WRITING GAUSS MAIN/RB
```

アセンブル作業を実行して、リロケータブルバイナリファイルを出力せよ
読み込みを行うカセットファイルは "GAUSS MAIN" である

出力リロケータブルバイナリファイル名は "GAUSS MAIN/RB" とする
(CASSETTEの場合)

```
PASS 4
FILENAME?GAUSS MAIN/RB [CR]
↓ RECORD.PLAY
WRITING GAUSS MAIN/RB
```

アセンブル作業を実行して、リロケータブルバイナリファイルを出力せよ
出力ファイル名は "GAUSS MAIN/RB" とする。
(RAMの場合)

—— "PASS" のコマンド待ちに、4を入力する。

—— PASS 1でアセンブル作業の方法をCASSETTEに選んでいる場合、システムは、一担RAMエリアにリロケータブルバイナリを作成するために "FILENAME?" と表示して読み込むべきソースファイル名の指定を待つ。

ファイルネームを指定し、[CR] キーを押すと、PLAYボタンを押すよう指示される。

PLAYボタンを押すと、システムは読み込むべきソースファイルを見つけ出してソースプログラムを読み込み、アセンブル作業を始める。

—— ソースファイルの読み込みは256バイトずつ行われる。

—— PASS 1でアセンブル作業の方法をRAMに選んでいる場合は、RAMエリアのソースプログラムをアセンブルし、リロケータブルバイナリを作成する。コマンドを与えて暫くの間、システムの応答が途切れるように見えるが、この間は一担ソースプログラムの全体をアセンブルしてRAMエリアにリロケータブルバイナリを構成している過程である。

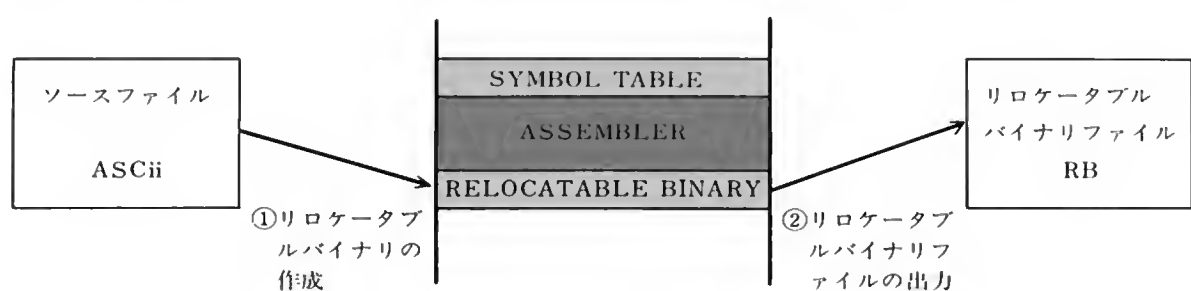
—— RAMエリア内に、リロケータブルバイナリが構成されると、システムは "FILENAME?" と表示して、出力するリロケータブルバイナリファイルのファイルネームの指定を待つ。

—— ファイルネームを指定し、[CR] キーを押すと、システムは、RECORDボタンとPLAYを押すよう指示する。

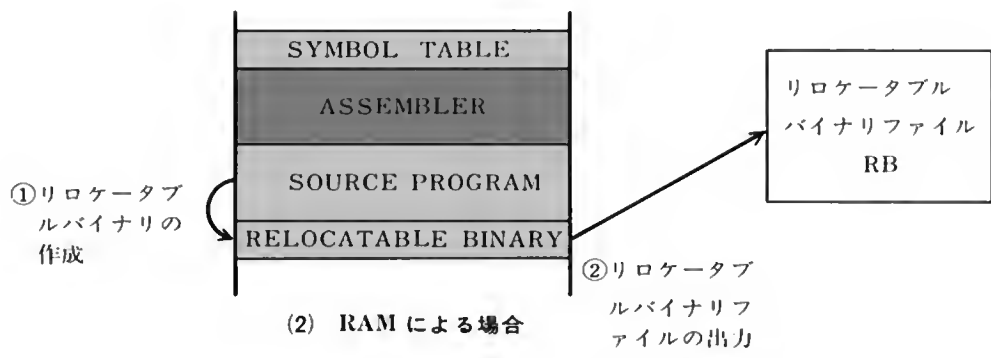
—— RECORDおよびPLAYボタンを押すと、リロケータブルバイナリにファイルネームを付けて、カセットファイルへ出力する。

—— コマンドを中止するには、[SHIFT] [BREAK] を押す。

—PASS 4 の実行形態を下図に示す。CASSETTE、RAMのいずれの方法をとる場合も、はじめに、ソースプログラムを読み込みアセンブルしてリロケートブルバイナリをRAMエリアに構成する過程があり、それが終わったら、リロケートブルバイナリをカセットファイルへ出力する過程が来る。



(1) CASSETTE による場合



(2) RAM による場合

〔参考〕

システムプログラムでは、アセンブリ語によるソースファイルから目的ファイルまで、数種類のファイルが用いられるので、ファイルの種類を示す記号をファイルネームに加えておくと便利である。

たとえば、エディタ出力のソースファイルには "ASCIi" または "SOURCE", アセンブラ出力のリロケートブルバイナリファイルには "RB" または "REL", ロード出力の目的プログラムには "OBJ" あるいは "BINARY", "SAVE", デバッガ出力のバイナリには "DEB" などという記号が考えられる。

第 3 章

TEXT EDITOR

SP-2201

mz-80k



3 — 1 テキストエディタ概要

テキストエディタはアセンブラソースファイルを作成する。あるいは、作成されたファイル形式のアスキーデータを入力して、修正、編集を行ない、更新されたアセンブラソースファイルを出力する。

データの修正、編集は、エディタとプログラマの対話形式によって実行される。

ソースプログラムは、アセンブリ言語、アセンブラ規約に従った文法で作成されなければならない。ソースプログラムはアスキーコードでコード化され、各行の区切りは、キャリジリターンコードである。ソースプログラムはEND文によって終了しなくてはならない。

修正、編集を行うための機能として、次のような作業を行うことができる。

挿入 (insertion)

削除 (deletion)

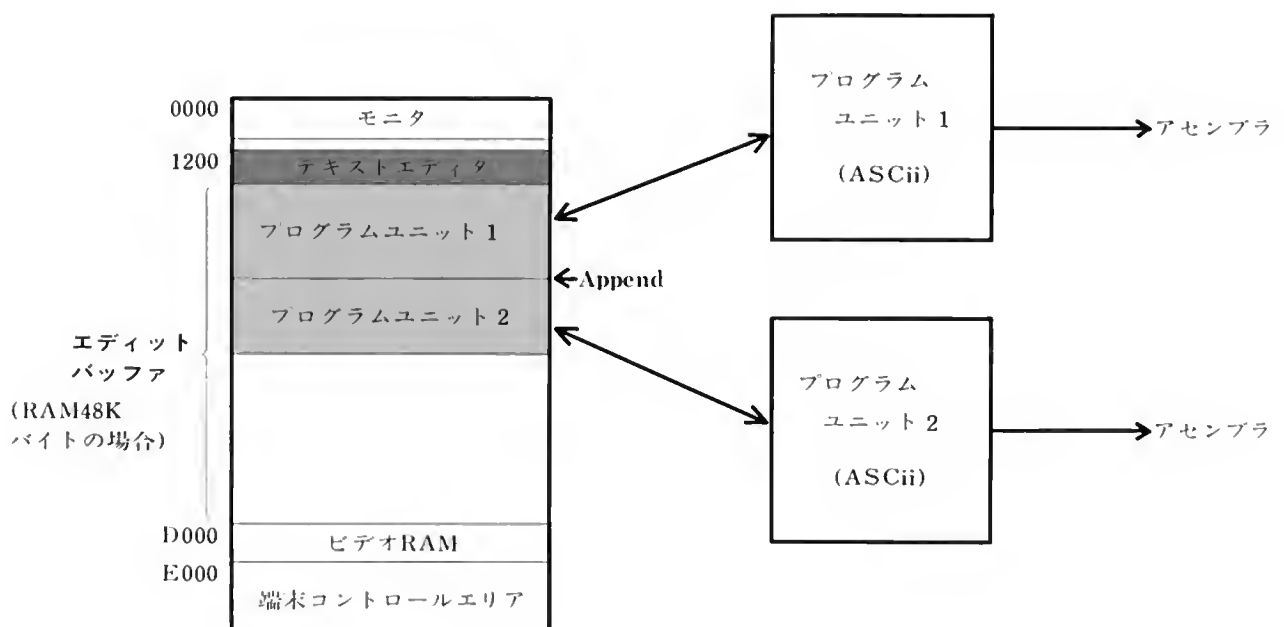
変更 (change)

エディットバッファ内に入力されたデータは、行 (line) と欄 (column) との2次元に構成され、各行には、エディットバッファの先頭より、順次番号が付され、これを、ラインナンバ (line number) と呼ぶ。

エディットバッファ内の修正箇所は、キャラクタポインタ (character pointer: 以下、CPと略す。) と呼ぶ修正子を基準として指定する。前記の、修正、編集を行うには、ラインナンバを参考にCPを移動させてコマンドの実行を行うことになる。

修正、編集の作業は、行または文字単位で行うことができる。また、文字列 (ストリング) 単位で、検索あるいは、文字列の入れ換えを行うことも可能である。

テキストエディタを用いる場合のメモリ構成と、利用形態を、下図に示す。テキストエディタのバイトサイズは、ほぼ4Kバイトである。



テキストエディタのコマンドには次のものがある。各コマンドは、デリミタ"❖"によって区切られ、コマンドの実行は、キャリジリターンによる。

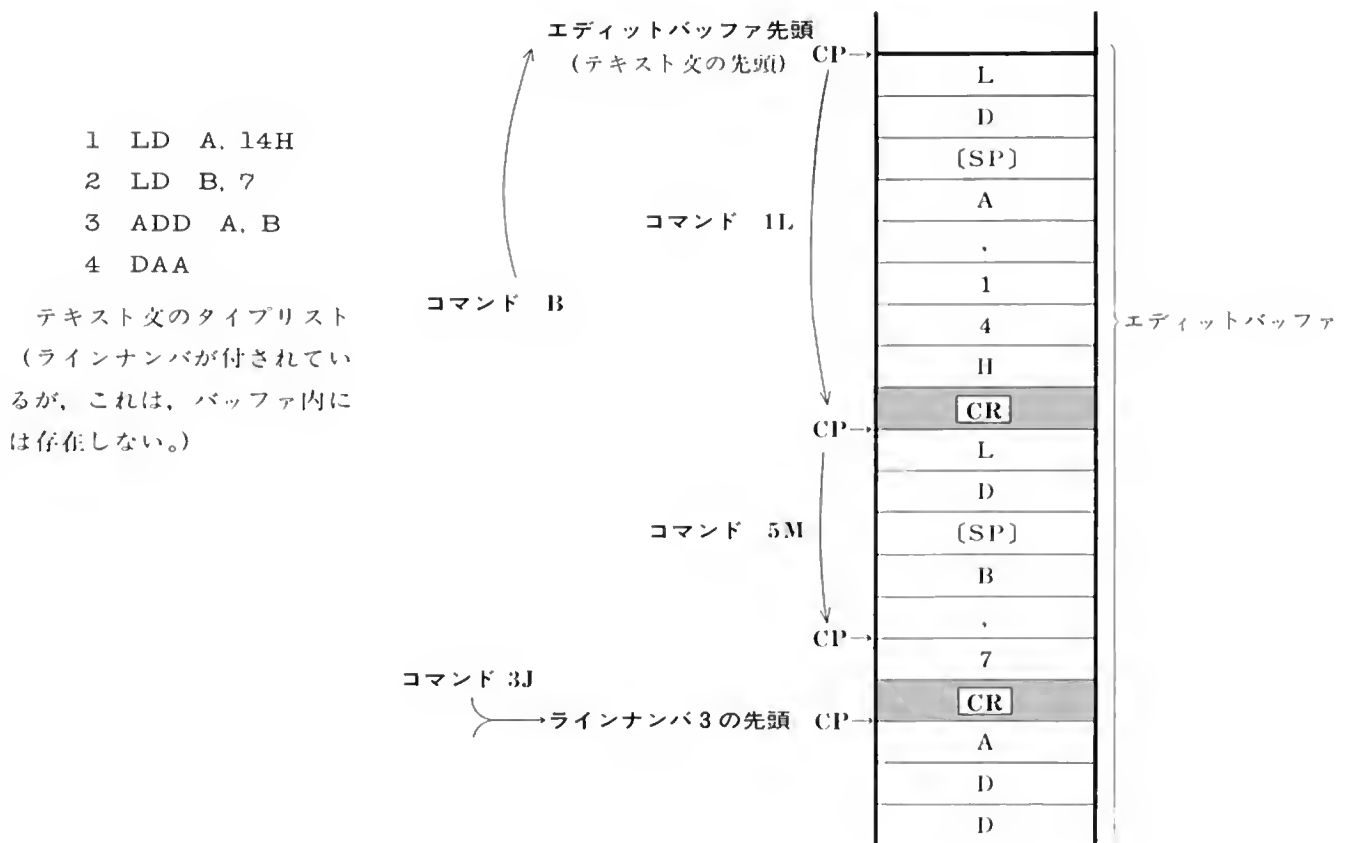
コマンドの種類	コマンド名	機 能
入力コマンド	R	エディットバッファ内をクリアし、filenameで指定する入力ファイルを入力する。実行後のCPは、エディットバッファの先頭に位置する。(Read file)
	A	エディットバッファの内容に、filenameで指定する入力ファイルをCPの位置から付加して入力する。実行後CPの位置は変化しない。(Append file)
タイプコマンド	T nT	エディットバッファの内容を全てタイプアウトする。CPは変化しない。(Type) CPの位置から、n行をタイプアウトする。
CP移動コマンド	B	CPの位置を、エディットバッファの先頭に移動する。(Begin)
	nJ	nで指定するラインナンバの先頭へCPを移動する。(Jump)
	nL	CPのある行から、n行目の行の先頭へCPを移動する。(Line)
	L	nLコマンドでn=0とした場合と同じで、CPを行の先頭へ移動する。
	nM	CPの位置を、n文字分だけ移動する。(Move)
	M	nMコマンドでn=0とした場合と同じで、CPは変化しない。
修正コマンド	Z	CPの位置を、エディットバッファ内のテキスト文の最後に移動する。
	C	CPの位置からバッファの終りに向って、あるストリングを検索し、それを他のストリングと交換する。CPは交換したストリングの後に置かれる。(Change)
	Q	CPの位置からバッファの終りに向って、連続してCコマンドを実行する。CPは最後に交換したストリングの後に置かれる。(Queue)
	I	CPの位置からストリングを入力する。CPは、挿入されたストリングの後に位置する。行を挿入するとラインナンバも更新される。
	nK	CPの位置から、n行をエディットバッファから消去する。CPは変化しない。(Kill)
	K	CPの位置から、エディットバッファの先頭に向って CR コードが1個検出されるまで、文字を消去する。 CR は消去されない。
	nD D	CPの位置から、n文字をエディットバッファから消去する。(Delete) nを指定しないと削除は行なわれず、CPも変化しない。
検索コマンド	S	CPの位置からバッファの終りに向って、あるストリングを検索する。実行後のCPは、検索されたストリングの後に位置する。(Search)
出力コマンド	W	エディットバッファの内容を、filenameで指定される出力ファイルに出力する。(Write)
比較コマンド	V	エディットバッファの内容と、filenameで指定される入力ファイルの内容とを比較する。実行後CPは変化しない。(Verify)
特殊コマンド	=	エディットバッファのすべての文字数を表示する。スペース、 CR も含む。
	.	現在のCPの位置するラインナンバを表示する。
	&	エディットバッファの内容をすべて消去する。
	#	プリンタへのリスティングのため、リストモードを切り替える。
	!	モニタへコントロールを移す。復帰はGOTOS1200 (cold start), GOTOS1260 (warm start)。

(テキストエディタの以上のコマンドは、ほぼ、日本ミニコンNOVA、エディタプログラムのコマンド形式とコンパチブルになっている。)

3—2 キャラクタポインタ(CP)とデリミタ

キャラクタポインタ (CP) は、エディットバッファ内のソースプログラムテキスト中の1箇所、ある隣り合った2つのキャラクタの境目あるいは、テキストの先頭か最終を指すものであり、1つのキャラクタを指すものではない。

エディットバッファ内に、次のようなテキストがある場合を考えて、CPの移動を説明する。



B コマンドは、エディットバッファの先頭へ、J コマンドは、ラインナンバで指定する行の先頭へ、L コマンドは、現在の行からn行目の行の先頭へCPを移動させる。"先頭"とは、前の行の終りを示す **CR** コードの後ろの境界線上である。

デリミタ (delimiter) は区切り記号 (セパレータともいう) として用いられる。即ち、1つのコマンドを与え終えたところなどで使用する。数個のコマンドをデリミタで区切って与え、**CR** をキー入力すると、コマンドが順番に実行されて行く。

I (Insert) コマンドでは、**CR** をソーステキストのキャラクタコードとして用いているので、コマンドの終りに必ず、デリミタを置く必要がある。

デリミタを用いて、上記テキスト文の、ラインナンバ3のADDをADCに変更するコマンド例を次に示す。

3J❖2M❖1D❖IC❖ **CR** または B❖CADD❖ADC **CR**

3 — 3 テキストエディタコマンド

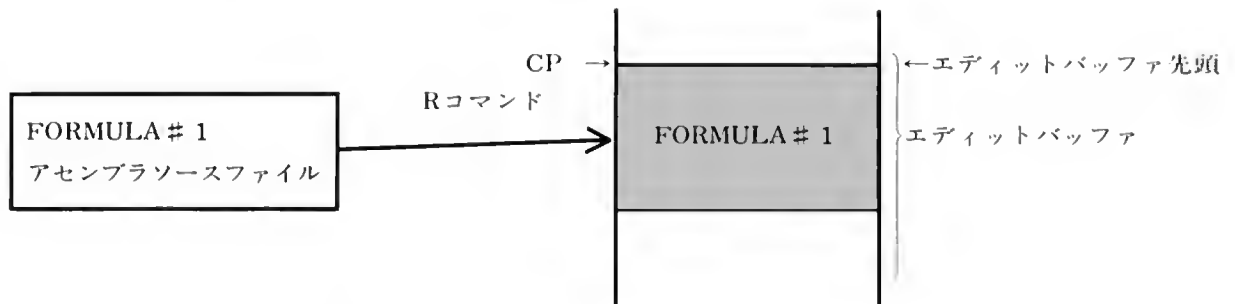
——入力コマンド——

R(Read file)コマンド

エディットバッファ内をクリアし、filenameで指定するアセンブラソースファイルを、エディットバッファの先頭から入力する。実行後のCPは、エディットバッファの先頭に位置する。

*RFORMULA #1 CR	ファイルネーム " FORMULA #1 " なるソースファイルを入力せよ
*R CR	最初に見つかったソースファイルを入力せよ

- " * " のコマンド待ちに R (Read file) コマンドを与える。
- 続けて、ファイルネームを指定する。
あるいは、ファイルネームを指定しない。
- **CR** を押すと、PLAYボタンを押すよう指示される。
- PLAYボタンを押すと、指定されたファイルを見つけ出し、読み込みを始める。
ファイルネームを指定していないと、最初に見つけ出したファイルを読み込む。
- 入力ファイルは、エディットバッファの先頭からストアされる。(下図)
- 読み込みが終了すると、" OK " が表示され、CPはエディットバッファの先頭に位置する。
- Rコマンドを中止する時は、**SHIFT** **BREAK** を押す。



- 読み込みの途中でエラーが生じた場合、" ERROR " を表示する。
- バッファがフルになった時には、" FULL BUFFER " のメッセージがある。この場合、入力ファイルは、途中までしか読み込まれないことになる。

A(Append file)コマンド

エディットバッファの内容に、filenameで指定する入力ファイルを、エディットバッファ内で、現在指定されるCPの位置から付加して入力する。実行後、CPの位置は変化しない。

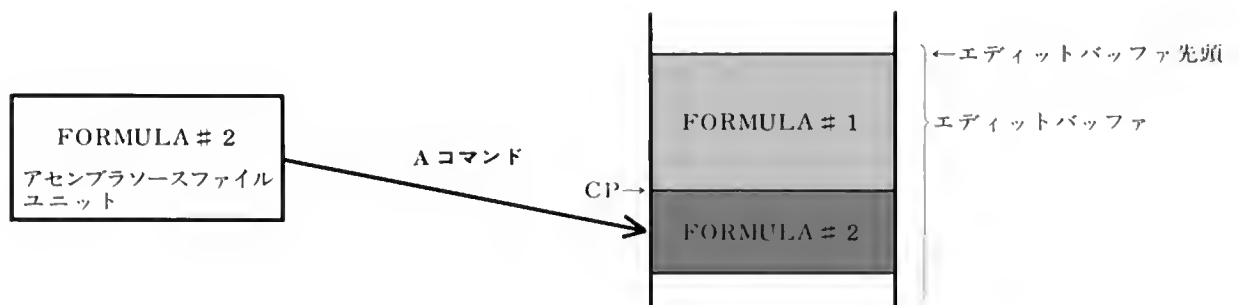
*A FORMULA #2 **CR**

ファイルネーム " FORMULA #2 " なるソースファイルを、CPの位置から付加せよ

*A **CR**

最初に見つかったソースファイルを、CPの位置から付加せよ

- " * " のコマンド待ちに A (Append file) コマンドを与える。
- 続けて、ファイルネームを指定する。
あるいは、ファイルネームを指定しない。
- **CR** を押すと、PLAYボタンを押すよう指示される。
- PLAYボタンを押すと、指定されたファイルを見つけ出し、読み込みを始める。
ファイルネームを指定していないと、最初に見つけ出したファイルを読み込む。
- 入力ファイルは、エディットバッファ内のCPの位置からストアされる。テキスト文の後ろに付加するには % コマンドによってCPをテキスト文の最後に置いておけばよい。下図は、テキスト文 " FORMULA #1 " の後ろに、入力ファイル " FORMULA # 2 " を付加するもようを示している。



- 読み込みが終了すると、 " OK " 表示され、CPは付加されたデータの先頭に位置する。
- A コマンドを中止する時は、 **SHIFT** **BREAK** を押す。
- 読み込みの途中でエラーが生じた場合、 " ERROR " を表示する。
- バッファがフルになった時には、 " FULL BUFFER " のメッセージがある。この場合、入力ファイルは、途中までしか読み込まれないことになる。従って全部を入力するには、バッファ内を編集し直し、CP の位置を、前へもって行く必要がある。

——タイプコマンド——

T (Type) コマンド

エディットバッファの内容を、ラインナンバを付けてタイプアウトする。バッファの内容を全てタイプアウトする場合と、CPの位置から指定行数だけタイプアウトする場合とがある。実行後CPの位置は、変化しない。

*T **CR** エディットバッファの内容を、ラインナンバを付けて全てタイプアウトせよ
*nT **CR** CPの位置から、n行をタイプアウトせよ (n=0の時は、上と同じ)

—— "＊" のコマンド待ちに、行数 n、T (Type) の順にコマンドを与える。

—— **CR** を押すと、タイプアウトが実行される。

—— 行数の指定で、特別の場合を次に示す。

n = 0 の時 Tと同機能

n < 0 の時 " ??? " のエラー表示がある

n ≥ m (mは現在のCPからバッファの終りまでの行数) の時m行のみのタイプアウト

—— CPが行の先頭でなく、行の中にある時は、nTコマンドは、CPの次の文字からタイプアウトするので、逆にCPの位置を知ることができる。

—— Tコマンドを中止するときは、**SHIFT** **BREAK** を押す。

—— 右の写真は、次のテキスト文について、タイプコマンドとCPの関係を示すものである。

```
1  START:ENT
2  LD  SP, START
3  CALL MSTP ;MUSIC STOP
4  CALL LETNL ;NEW LINE
5  END
```

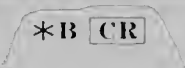



```
#T
1  START:ENT
2  LD  SP, START
3  CALL MSTP ;MUSIC STOP
4  CALL LETNL ;NEW LINE
5  END
#3J#2T
3  CALL MSTP ;MUSIC STOP
4  CALL LETNL ;NEW LINE
5  END
#10M#2T
3  CALL MSTP ;MUSIC STOP
4  CALL LETNL ;NEW LINE
5  END
```


n の値が65535を越えると、" LARGE "のエラー表示がなされる。

——CP移動コマンド——

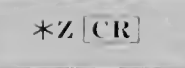

B(Begin) コマンド

 *B 


CPの位置を、エディットバッファの先頭へ移動せよ

- " * " のコマンド待ちに B (Begin) コマンドを与える。
-  を押す。
- B コマンドが実行されて、CPはエディットバッファの先頭に位置する。
- nBとしても同機能である。

Zコマンド

 *Z 


CPの位置を、エディットバッファ内のテキスト文の最後へ移動せよ

- " * " のコマンド待ちに Z コマンドを与える。
-  を押す。
- Z コマンドが実行されて、CPはエディットバッファ内のテキスト文の最後（最後の文字の次）に位置する。
- nZとしても同機能である。

J(Jump) コマンド

 *nJ 

CPの位置を、ラインナンバ n で示す行の先頭へ移動せよ

- " * " のコマンド待ちに、ラインナンバ n、J (Jump) の順にコマンドを与える。
-  を押す。
- nJ コマンドが実行されて、CPはラインナンバ n で示す行の先頭に位置する。
- ラインナンバの指定で、特別の場合を次に示す。

n = 0 または 1、あるいは無指定の時 Bと同機能

n < 0 の時 " ??? " のエラー表示がある。

n ≥ m (mはテキスト文の全行数) の時、 Zと同機能

L(Line) コマンド

CPを、行 (Line) 単位で前後に移動させるコマンドである。実行後CPは、指定された行の先頭に位置する。

*nL [CR]	CPのある行からn行目にあたる行の先頭へCPを移動せよ
*L [CR]	CPのある行の先頭へCPを移動せよ

—— "＊" のコマンド待ちに、**行数 n**、**L (Line)** の順にコマンドを与える。

—— **[CR]** を押す。

—— nL コマンドが実行されてCPは、指定された行の先頭に位置する。

—— 行数の指定で特別の場合を次に示す。

n = 0 の時、L と同機能

n ≥ m (m は、現在のCPの位置から、テキスト文の終りまでの行数) の時、Z と同機能

n < 0 の時、CPのある行から、バッファの先頭に向かって |n| 行移動する。

|n| ≥ l - 1 (l は現在のCPのある行のラインナンバ) の時、B と同機能

M(Move) コマンド

CPを、文字単位で前後に移動させるコマンドである。スペース、キャリジリターンも、それぞれ1文字として数える。(ラインナンバは含まれない)

*nM [CR]	CPのある位置 (文字と文字の間) からn文字分だけCPを移動せよ
-----------------	-----------------------------------

—— "＊" のコマンド待ちに、**文字数 n**、**M (Move)** の順にコマンドを与える。

—— **[CR]** を押す。

—— nM コマンドが実行されてCPは、指定された文字分だけ離れた位置 (文字と文字の間) へ移動する。

—— n < 0 の時、CPは |n| 文字分だけ、バッファの先頭方向へ移動する。

—— n = 0 または n が無指定の時は、CPは変化しない。

——修正コマンド——

C(Change)コマンド

エディットバッファ内のあるストリングを、他のストリングに代えるコマンドである。ストリングの検索はC Pの位置から、バッファの終りへ向って行なわれ、ストリングの交換を1回だけ実行し、CPは置き換えたストリングの後ろに位置する。

*Cstring 1❖string 2 [CR]

現在のCPの位置からバッファの終りへ向ってstring 1で指定される文字列を検索し、それが見つかったら、string 2に置きかえよ

*Cstring 1 [CR]

[上記と異なる点は、string 1の消去にある]

- " *" のコマンド待ちにC (Change) コマンドを与える。
- 検索するストリングをキー入力し、デリミタを置く。
- 置きかえるストリングをキー入力する。
- [CR] を押すと、ストリングの検索を始める。ストリングの置き換えは1回だけ行なわれ、置き換えられたストリングを含む行を表示する。CPは、置き換えられたストリングの後ろに位置する。
- 検索するストリングが見つからなかった場合は、" NOT FOUND " のメッセージ表示がなされ、CPはエディットバッファの先頭へ位置する。

Q(Queue)コマンド

Cコマンドと異なる点は、ストリングの検索、置き換えが行なわれたあとも、連続的にCコマンドを実行する点である。実行後、CPは最後に置き換えたストリングの後ろに位置する。

*Qstring 1❖string 2 [CR] 現在のCPの位置から連続的にCコマンドを実行せよ

*Qstring 1 [CR]

[上記と異なる点は、string 1の消去にある]

- " *" のコマンド待ちにQ (Queue) コマンドを与える。
- あとの操作は、Cコマンドと同じである。置き換えられた行は全て表示する。

- 右の写真は、下に示すテキストについて、Qコマンドを実行したもようを示す。

```
1 LD BC, (TEMPO)
2 LD (TEMPO), DE
3 JP 1200H
4 TEMPO: DEFS 2
```

```
*T
1 LD BC, (TEMPO)
2 LD (TEMPO), DE
3 JP 1200H
4 TEMPO: DEFS 2
*BQ TEMPO, BUFFER
1 LD BC, (BUFFER)
2 LD (BUFFER), DE
4 BUFFER: DEFS 2
*T
1 LD BC, (BUFFER)
2 LD (BUFFER), DE
3 JP 1200H
4 BUFFER: DEFS 2
*
```

I (Insert) コマンド

CPの位置から、キー入力されたストリングをエディットバッファに挿入する。キャリジリターンを挿入すると、テレビ画面上でも、行替えが行われる。

テキスト文中に、新しい行を挿入する場合は、エディットバッファ内の各行に付けられたラインナンバも更新される。実行後のCPは挿入された文字列の後に位置する。

*Istring	※	CR	CPの位置にstringを挿入せよ
*Istring1		CR	CPの位置から、string1, string2, string3の各行を挿入せよ
string2		CR	
string3		CR	
※		CR	[Iコマンド内では CR は、バッファに挿入されるコードの1つであるから デリミタを置いた後 CR でコマンドを実行させる]

—— " * " のコマンド待ちに、I (Insert) コマンドを与える。

—— 直ちに挿入可能な状態となり、ストリングの入力待ちとなる。

—— CPの位置から、キー入力されたアスキーコードを挿入する。従って、CPの後ろに置かれていたストリングは、挿入が行われる度に、エディットバッファの後ろに向って、順送りされることになる。

—— CR がキー入力されると、行の区切り記号として CR コードがストアされる。

—— ストリングの挿入が終わったら、デリミタ ※ を置く。

—— CR を押すと、I コマンドが実行される。

—— 次に示すテキスト文への挿入操作例を、右の写真に示す。

テキスト文

```
1  START:ENT
2  LD  S, START
3  CALL MSTP ;MUSIC STOP
4  CALL XTEMP ;SET TEMPO
5  END
```

ラインナンバ3と4の間に

```
LD A, 5 ;TEMPO 5
```

を挿入する。

```
#T
1 START:ENT
2 LD SP, START
3 CALL MSTP ;MUSIC STOP
4 CALL XTEMP ;SET TEMPO
5 END
#4JILD A, 5 ;TEMPO 5
#T
1 START:ENT
2 LD SP, START
3 CALL MSTP ;MUSIC STOP
4 LD A, 5 ;TEMPO 5
5 CALL XTEMP ;SET TEMPO
6 END
#
```

K (Kill) コマンド

CP の位置から、 n で指定する行数を、エディットバッファから消去する。

- | | |
|--|---|
| * n K CR | CP の位置を基準として、 n により指定される行数をエディットバッファより消去せよ。
CP が行の中間に位置しているときは、その行の CP の前または後ろにある文字は消去しない。 $(n$ の値が正か負かで異なる) |
| * K CR | CP の位置を基準として、エディットバッファの前に向かって CR が 1 個検出されるまですべての文字を消去せよ。 CR は消去されない。 |

—— " * " のコマンド待ちに、行数 n 、K (Kill) の順にコマンドを与える。

—— CR を押すとコマンドが実行される。

—— コマンドの実行は、 n の値によってそれぞれ次のようになる。

- | | |
|-----------------------|---|
| $n > 0$ の場合 | 現在の CP の位置を基準として、エディットバッファの後ろに向かって CR コードが n 個検出されるまで、すべての文字をエディットバッファより消去する。
CR コードも削除されるので、 n 個目の CR が削除された時コマンドの実行を終了することになる。 |
| $n < 0$ の場合 | 現在の CP の位置を基準として、エディットバッファの前に向かって CR コードが $ n + 1$ 個検出されるまで、すべての文字をエディットバッファより削除する。
$ n + 1$ 個目の CR は削除されない。 |
| $n = 0$ または
無指定の場合 | 現在の CP の位置を基準として、左方向に CR が 1 個検出されるまでのすべての文字をエディットバッファより消去する。従って、CP の位置から、その行の先頭までの文字がすべて削除される。 CR は消去されない。 |

—— 行の削除が行なわれたら、ラインナンバも更新される。

—— 実行後、CP の位置は変化しない。

—— 右の写真は、下のテキスト文について、K コマンドを実行したもようを示す。(このテキスト文は、コマンドの実行をわかり易くするためのものである)

```
1 AABBC
2 DDEEFF
3 GGHHII
4 JJKKLL
```



D (Delete) コマンド

CP の位置から、n で指定する文字数を、エディットバッファから消去する。

- * nD [CR] CP の位置を基準として、n により指定される文字数をエディットバッファより消去せよ。
[CR] も文字数に数える。
- * D [CR] (削除は行われず、CP も変化しない)

—— " *" のコマンド待ちに、文字数 n、D (Delete) の順にコマンドを与える。

—— [CR] を押すとコマンドが実行される。

—— コマンドの実行は、n の値によってそれぞれ次のようになる。

- n > 0 の場合 現在の CP の位置を基準として、エディットバッファの後ろに向かって、n 個の文字をエディットバッファより削除する。
[CR] コードも 1 文字に数える。
- n < 0 の場合 現存の CP の位置を基準として、エディットバッファの先頭に向かって、| n | 個の文字をエディットバッファより削除する。
[CR] コードも 1 文字に数える。
- n = 0 または 削除は行われない。
無指定の場合

—— 文字の削除が、1 行を越えた場合ラインナンバも更新される。

—— 実行後 CP の位置は変化しない。

右の写真は、下のテキスト文について、D コマンドを実行したもようを示す。(このテキスト文は、コマンドの実行をわかり易くするためのもので、アセンブリ語としての意味はもたない。)

- 1 ABCD
- 2 EFGH
- 3 IJKL
- 4 MNOP



—— 検索コマンド ——

S (Search) コマンド

エディットバッファ内のテキスト文中のストリングを見つけ出す。

* S string **CR**

現在の CP の位置からエディットバッファの終りに向って string で示される文字列を検索せよ
string が検索されたら、CP をその直後に位置させよ

—— " *" のコマンド待ちに、S (Search) コマンドを与える。

—— 検索すべき string を入力する。

—— **CR** を押すと、検索が実行される。

—— 検索は、CP の位置からエディットバッファの終りに向ってなされる。

—— string が見つかったら、その string を含む行を表示して、CP は、string の直後に置かれる。

—— string が見つからなかったら、" NOT FOUND " の表示がなされ、CP はエディットバッファの先頭に置かれる。

—— 右の写真は、下のテキスト文について string " LETNL " を検索したもようを示している。

S コマンドのあとに、" LETNL " を含む行が表示されるが、次の 2T コマンドで、CP が、この string の後に位置していることがわかる。

```
1 START:ENT
2 LD SP, START
3 CALL MSTP ;MUSIC STOP
4 CALL LETNL ;NEW LINE
5 LD A, 04H ;TEMPO<--4
6 CALL XTEMP
7 END
```

```
#T
1 START:ENT
2 LD SP, START
3 CALL MSTP ;MUSIC STOP
4 CALL LETNL ;NEW LINE
5 LD A, 04H
6 CALL XTEMP ;TEMPO<--4
7 END
#B SLETNL
4 CALL LETNL ;NEW LINE
#2T
4 ;NEW LINE
5 LD A, 04H
#
```

——出力コマンド——

W (Write) コマンド

エディットバッファの内容を、filename で指定される出力ファイルに出力する。出力されるテキストは、CP の位置に関係なく、バッファ内の全テキストである。

* WFORMULA # 3 **CR**

エディットバッファの内容に " FORMULA # 3 " なる filename を指定して、出力ファイルに出力せよ

* W **CR**

エディットバッファの内容を、filename を指定せずに出力ファイルに出力せよ

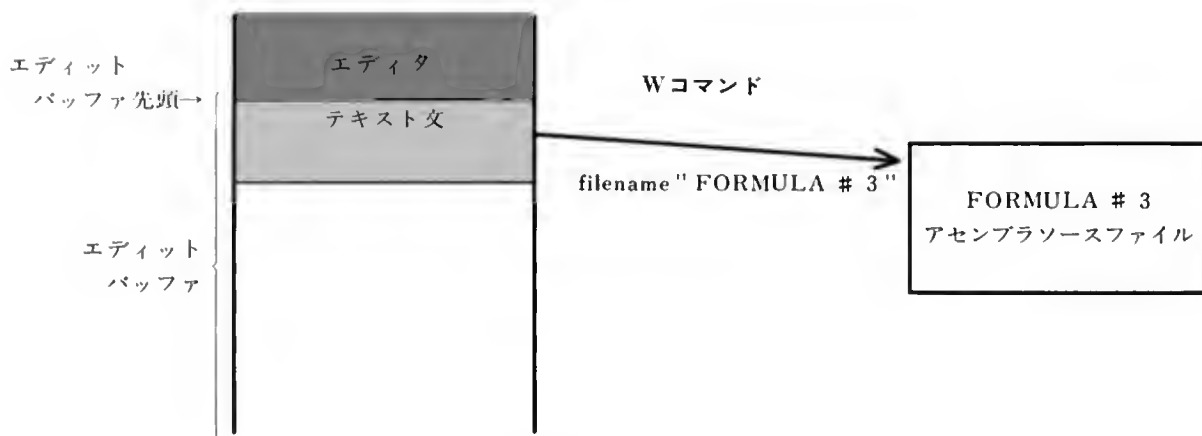
—— " * " のコマンド待ちに、W (Write) コマンドを与える。

—— filename を指定する。

—— **CR** を押すと、RECORD ボタンと、PLAY ボタンを押すよう指示される。

—— RECORD および PLAY ボタンを押すと、テキストのファイルの出力が開始される。

—— 出力ファイルへの出力が終わると、" OK " 表示される。作成されたファイルは、アセンブラソースファイルである。



—— W コマンドを中止する時は、**SHIFT** **BREAK** を押す。

—— 出力ファイルへの書き出しの途中でエラーが生じた場合、" ERROR " を表示する。

—— W コマンドの実行前後で、CP は変化しない。

——比較コマンド——

V (Verify) コマンド

エディットバッファの内容と、filename で指定されるファイルの内容とを比較する。

＊VFORMULA # 3 **CR**

エディットバッファの内容と、ファイル" FORMULA # 3 "の内容とを比較せよ

＊V **CR**

エディットバッファの内容と、入力ファイルの内容とを比較せよ

—— "＊" のコマンド待ちに、V (Verify) コマンドを与える。

—— 比較すべき入力ファイルの、filename を指定する。

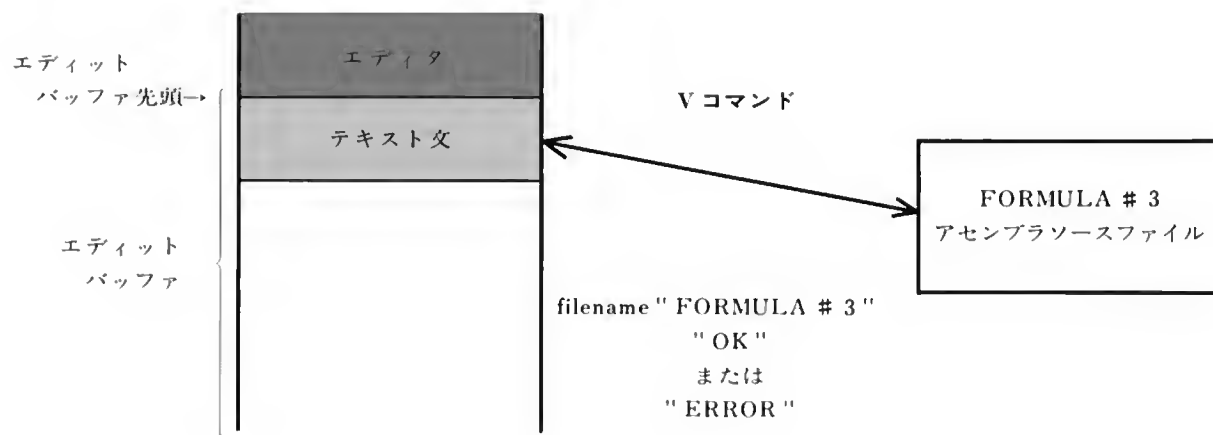
—— **CR** を押すと、PLAY ボタンを押すように指示される。

—— PLAY ボタンを押すと、filename に指定された該当ファイルを見つけ出し、見つかったら比較 (Verify) を開始する。

filename が指定されなかった場合は、はじめに見つかったファイルと、バッファの内容とを比較することになる。

—— 入力ファイルとエディットバッファの内容が同一であれば、" OK " と表示され、相違があれば、" ERROR " 表示がなされる。

—— 実行後 CP の位置は変化しない。



—— 特殊コマンド ——

= コマンド

* = **[CR]** エディットバッファにあるすべての文字の総数を表示せよ
スペースおよび **[CR]** も 1 文字として数えよ

—— " * " のコマンド待ちに、 = (equal) コマンドを与える。

—— **[CR]** を押すと、エディットバッファ内のすべての文字数が表示される。

. コマンド

* . **[CR]** 現在の CP の位置する行のラインナンバを表示せよ

—— " * " のコマンド待ちに、 . (period) コマンドを与える。

—— **[CR]** を押すと、CP のある行のラインナンバが表示される。

& コマンド

* & **[CR]** 現在のエディットバッファの内容をすべてクリアせよ

—— " * " のコマンド待ちに、 & (ampersand) コマンドを与える。

—— **[CR]** を押すと、エディットバッファの内容がすべてクリアされる。

コマンド

* # [CR] プリンタリストモードを切り替えよ

—— " * " のコマンド待ちに、 # (sharp mark) コマンドを与える。

—— [CR] を押すと、プリンタリストモードが切り替わる。

—— テキストエディタの始動時は、リストモードは、disable である。# コマンドを 1 回入力すると、リストモードは enable、更に入力すると disable と順次、モードが反転することになる。

—— 下図は、オプションの放電プリンタへ、プリンタモードが enable の時に T コマンドによるリストを行なった印字例である。

```
1 .  
2 .  
3 . EDITOR LIST SAMPLE  
4 .  
5 REL 1200H  
6 START:ENT  
7 MAIN1:ENT  
8 LD SP,START ;INITIAL STACK POINTER  
9 CALL MSTP ;MUSIC STOP  
10 LD A,5  
11 CALL XTEMP ;SET TEMPO TO 5  
12 CALL CLTBL ;CLEAR TABLE  
13 XOA A  
14 LD (?TABP),A ;INITIAL I/O #1  
15 MSTP:EQU 0047H ;MUSIC STOP (MONITOR)  
16 ?TABP:DEFS 1  
17 END
```

! コマンド

* ! [CR] モニタへコントロールを移せ

—— " * " のコマンド待ちに、 ! (exclamation mark) コマンドを与える。

—— [CR] を押すと、コントロールはモニタへ戻る。

—— モニタからテキストエディタへ戻るには、次の 2 通りの方法がある。

GOTO\$1200 [CR] エディットバッファの内容はクリアされる。 (COLD START)

GOTO\$1260 [CR] エディットバッファの内容はクリアされない。 (WARM START)

EDITOR-ASSEMBLER (SYSTEM PROGRAMのOPTIONとして別売) というソフトは、文字通り、TEXT EDITORとASSEMBLERの両者を1本のシステムプログラムとしてまとめたものである。エディタおよびアセンブラの基本機能は前記のエディタSP-2201、アセンブラSP-2101とはほぼ同様であるが、EDITOR-ASSEMBLERでのバージョンナンバーは次のようになっている。

EDITOR-ASSEMBLER'S ASSEMBLER SP-2102

EDITOR-ASSEMBLER'S EDITOR SP-2202

エディタとアセンブラとは、それぞれ "X" コマンドで他へ移ることができる。このように、TEXT EDITORとASSEMBLERをまとめたのは、テープの掛け替えの煩わしさを省くためのものである。ソースプログラムのバイトサイズがそう大きくない場合、或いは、プログラムを作成し始める場合には、このソフトを用いて、エディット作業、アセンブル作業を続けて即座に行ない、アセンブルリストを検討しながら、ソースプログラム中の誤りをすぐに直すことができる。

EDITOR-ASSEMBLERのバイトサイズは、アセンブラシンボルテーブル (1200~2100) の4Kバイトを合わせて約14Kバイトである。

下の写真は、モニタで "EDITOR-ASSEMBLER" をロードし、はじめに走り出した TEXT EDITOR SP-2202で、3行のテキスト文を作成し、Xコマンドで、ASSEMBLER SP-2102へ移るもようと、逆に、ASSEMBLERのPASSからXコマンドで、エディタへ戻りTコマンドを行ったもようが示されている。

```

** MONITOR SP-1002 **
*LOAD
↓
*PLAY
LOAD EDITOR-ASSEMBLER
** TEXT EDITOR SP-2202 **
34071 BYTES
*ISTART:ENT
LD SP,START
END
*
*X
PASS

```

```

PASS X
*T
1 START:ENT
2 LD SP,START
3 END
*

```

モニタでEDITOR-ASSEMBLERをロードしている。

エディットバッファのバイト数が示されている。(写真ではRAMが最大48Kバイトの場合を示している)

エディタの "I" コマンドで3行のテキスト文を作成している。

エディタの "X" コマンドでアセンブラへ移り、PASSの入力待ちの状態となっている。

アセンブラのPASSで "X" コマンドを与えることにより、エディタへ移り、エディタの "*" のコマンド待ちとなる。

"T" コマンドを与えて、タイプアウトをした例である。

第 4 章

RELOCATABLE LOADER SP-2301

mz-80k



4—1 リロケータブルローダ概要

リロケータブルローダは、アセンブラ出力のリロケータブルバイナリファイルを入力し、目的プログラム（アブソリュートバイナリファイル）を出力する。

リロケータブルバイナリファイルというのは、CPUが直接に実行できるプログラムではなく、プログラムをリロケータブルな状態として保持するための、各種の情報が載せられているファイルである。

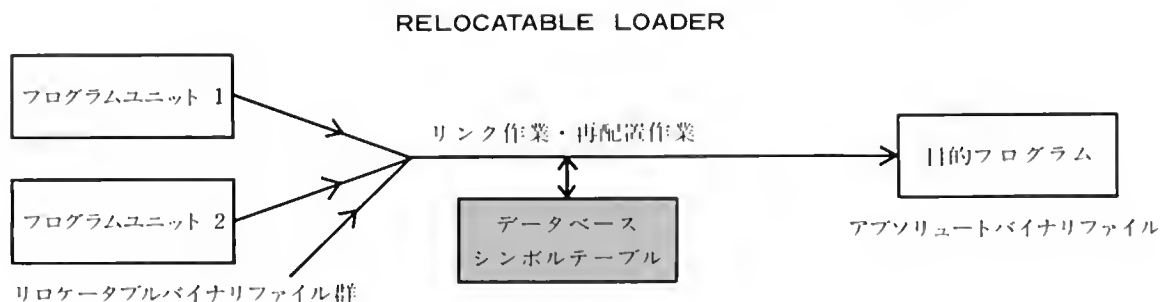
さらに、複数のプログラムユニットをリンクするために宣言されたグローバルシンボルは、そのままアスキーコードの形で、ファイルに登録されている。

リロケータブルローダは、これらの情報を受けとり、プログラマの指定するアセンブルバイアスを、相対アドレスに付加して、目的プログラムをなす機械語プログラムをローディングエリア内に構成して行く。

複数のリロケータブルバイナリユニットを入力する場合は、最初のプログラムでアセンブルバイアスを指定すれば、次からのプログラムユニットは、Nコマンドで前のプログラムに付加（Append）される形で、リンケージロードが行われる。

ローディングを行う、実際のメモリエリアは、アセンブルバイアスと同時に、プログラマが指定することになるが、そのローディングエリアは、一時的に使用されるメモリであり、一般に、目的プログラムのアドレス形式と一致しない。従って、リロケータブルローダは、実行コマンドを持たない。

目的プログラム(アブソリュートバイナリファイル)の出力は、実行アドレス(execute address)とデータアドレス(data address)を指定することにより行われる。ローダでのバイアスの考え方は後に解説される。



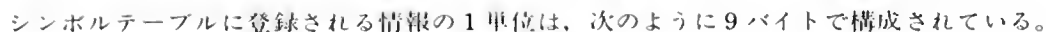
コマンド名	機 能
L (relocate load)	アセンブルバイアスとローディングアドレスを指定し、ローディングを開始する。
N (Next file)	前のプログラムに付加して次のプログラムをリンキングロードする。
H (Height)	現在のアセンブルバイアスとローディングアドレスの値を表示する。
T (Table dump)	シンボルテーブルの内容を表示する。
S (Save)	実行アドレスとデータアドレスを指定し、目的プログラムファイルを出力する。
V (Verify)	S コマンドで出力された目的プログラムファイルをメモリの内容と比較する。
* (clear table)	シンボルテーブルをクリアし、アセンブルバイアスとローディングアドレスを0000にする。
# (change printer mode)	プリンタ出力モードを切り替える。
! (goto monitor)	モニタへコントロールを移す。

リロケータブルロードおよびシンボリックデバッグで“シンボル”と呼ばれる情報とは、ソースプログラムでグローバル宣言のなされたラベルシンボルのことである。即ち、擬似命令ENTまたはEQUで定義されたラベルシンボルのことであり、これらはプログラムのリンクを行うために、アセンブラ出力のリロケータブルバイナリファイル中に情報としてそのまま残されているのである。

ローダは、リロケータブルバイナリファイルを入力しながら、ラベルシンボルを、シンボルテーブルにストアして行く。シンボルテーブルはローダのLINK AREA内の後部に置く。その先頭アドレスは16進の上位2桁をプログラマが指定する。たとえば、

とすると、シンボルテーブルの先頭アドレスは、8000H番地に設定される。

下の写真は、リロケータブルロードのイニシャライズの画面を示しており、右図は、メモリマップの構成を示している。



4-2 ロードのバイアスとアドレスの考え方

リンケージローダおよびシンボリックデバッガを使用する場合、プログラマは、シンボルテーブルのアドレス以外に、4つのアドレスを指定しなくてはならない。即ち、リロケータブルバイナリファイルを入力する際の、アセンブルバイアスとローディングアドレス、アブソリュートバイナリファイルを出力する際の、実行アドレスとデータアドレスである。

これらのアドレスによって、目的プログラムの形式を決めることになるが、それぞれを全く任意に指定することはできず、相互の関連に留意しなくてはならない。特に、REL命令を使用している場合は注意が必要である。

以下、順をおってこれらのバイアスの考え方を説明する。

ASSEMBLE BIAS

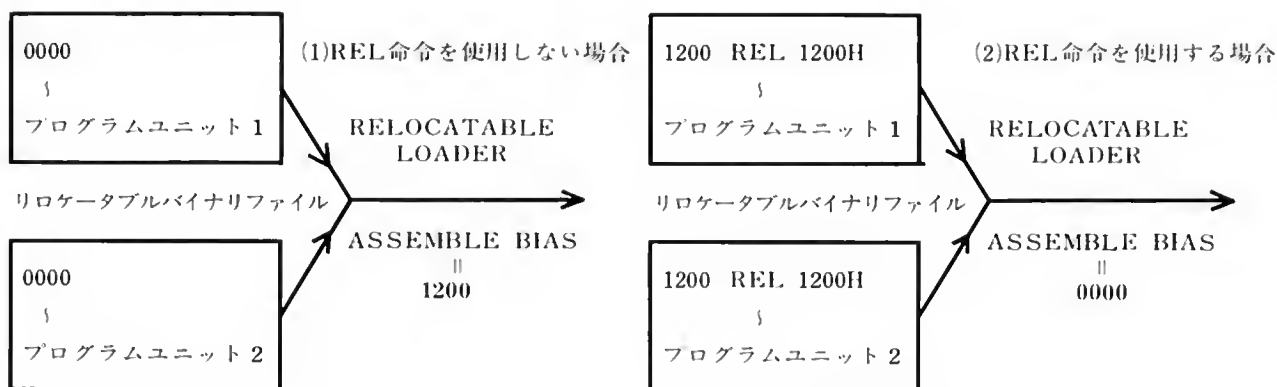
アセンブルバイアスは、目的プログラムのアドレス形式を、アブソリュート形式に決定するためのバイアスである。指定されたバイアスは、リロケータブルバイナリファイル中の相対アドレスに加算されることになる。

今、目的プログラムを、1200H番地からスタートする形式にする場合を考える。この場合、REL命令を用いなくてアセンブル作業を進める方法と、RELを使う場合と、2通りの進め方がある。RELを使う場合は、かなり、複雑になって来るので注意が必要である。

擬似命令RELを使わない時は、目的プログラムのスタートアドレス、今の場合1200Hを、ローダのアセンブルバイアスとして指定すればよい。即ち、アセンブラ出力のリロケータブルバイナリの相対アドレス形式は、いつも、0000Hをスタートアドレスとする形式となっているからである。

擬似命令RELを使って、すでに1200H番地からスタートする形式のリロケータブルバイナリファイルを得ている場合には、ローダのアセンブルバイアスは、0000に指定しなくてはならない。（それを、1200と指定すると、絶対アドレス形式は2400H番地からスタートする形式となる。）

複数のプログラムユニットをリンクする場合は、全てのプログラムユニットの、相対アドレス形式が一致していなければならない。たとえば、REL 1200Hをユニット1で使用したなら、他の全てのユニットの冒頭にもREL 1200Hが置かれている必要がある。



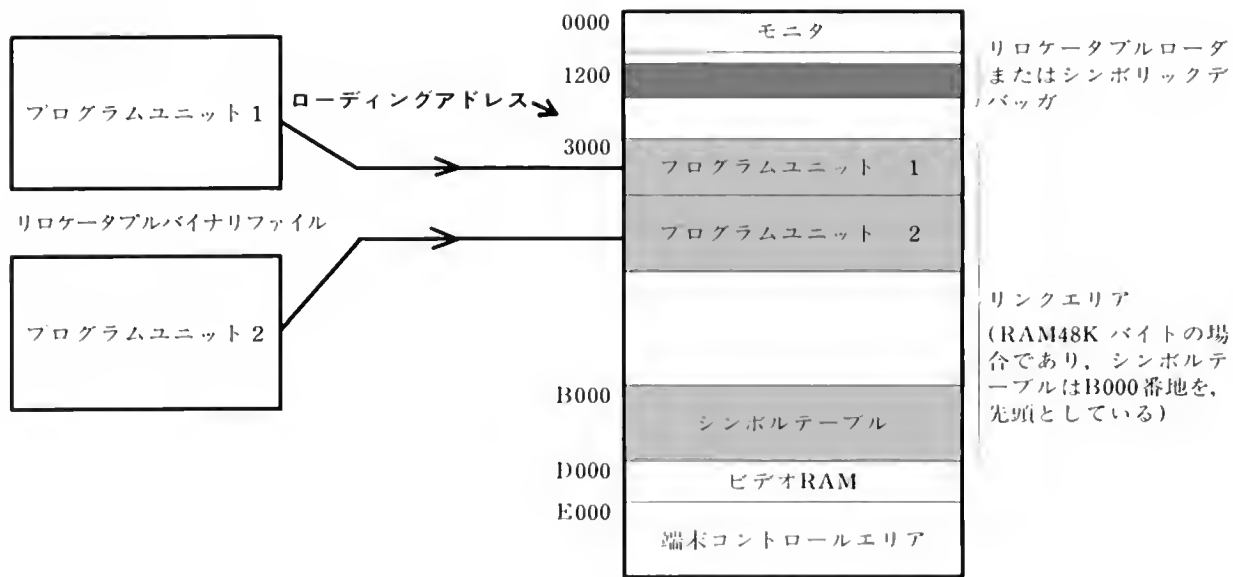
LOADING ADDRESS

ローディングアドレスは、リンクエリア内で、リロケートブルバイナリファイルを入力する先頭アドレスを指定するものである。

リロケートブルロードは、直接実行するコマンドを持っていない。すなわち、メモリ内に構成される、アブソリュートプログラムは、一般に、ストアされているメモリアドレスと形式が一致しないからである。従って、メモリ領域は一時的にストアされる領域であるから、任意のエリアを使用でき、ローディングアドレスも任意に、選ぶことが可能である。

一方、シンボリックデバッグは、直接実行するコマンドを持っており、アブソリュートプログラムは、直接実行可能でなくてはならず、アセンブルバイアスを指定すれば、ローディングアドレスとして指定すべき値が必然的に決まって来る。

次に、リロケートブルロードまたはシンボリックデバッグで、ローディングアドレスを3000と指定した時のローディング状態のもようを示す。



〔問〕 リロケートブルバイナリファイルが2つある。いずれも、ソースプログラムでRELを使っていない。シンボリックデバッグでリンキングロードを行なうのに、アセンブルバイアスを3000に指定して、アブソリュートプログラムを作成しデバッグを行なう。この場合、ローディングアドレスの値は、どれだけとしなくてはならないか。

答……………3000としなくてはならない。

〔問〕 今の2つのユニットが、いずれも先頭にREL 1200を使っており、アセンブルバイアスを、2000としたとすると、ローディングアドレスは、どれだけとしなくてはならないか。

答……………3200としなくてはならない。

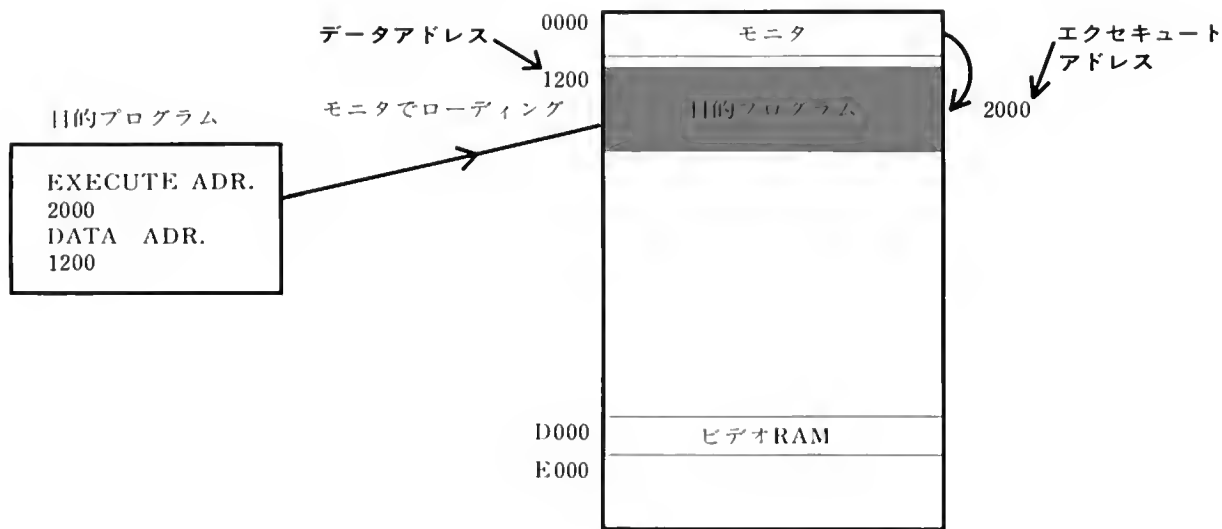
EXECUTE ADDRESS と DATA ADDRESS

エクセキュータアドレス（実行アドレス）とデータアドレスは、リロケータブルローダで目的プログラムファイルを出力する際に指定しなくてはならない。指定の際の任意性はなく、既にメモリ内に構成されている、アブソリュートプログラムの形式に従った値でなくてはならない。

これらのアドレスは、目的プログラムファイル（アブソリュートバイナリファイル、またはセーブモードオブジェクトファイルともいう）の中に、インフォメーションデータとして登録されるものである。

目的プログラムを、モニタでロードする場合、ロードされるアドレスの先頭は、データアドレスで指定される。ローディング終了後、プログラムカウンタのとり値は、エクセキュータアドレスで指定される。下図は、データアドレス1200、エクセキュータアドレス2000の目的プログラムのローディングとコントロールの移行のようが示されている。

目的プログラムを、モニタでなく、シンボリックデバッガまたはマシンランゲージのYコマンドでローディングを行う場合、或いは、BASICとリンクを行なう場合は、エクセキュータアドレスは無視され、ローディング終了後も、そのシステムプログラム中にコントロールは残る。従って、この場合は、そのシステムプログラムの実行コマンドを用いて、プログラムのコントロールを、エクセキュータアドレスへ移してやる必要がある。



4 — 3 リロケートブルロードコマンド

L (relocate Load) コマンド

ローダのリンクエリア内に、アセンブラ出力のリロケートブルバイナリファイルを入力する。ローダは、目的ファイル（アブソリュートバイナリファイル）を作成するためのものであり、そのアブソリュート形式は、このコマンドでアセンブルバイアスを指定することにより決められる。

* LL 1200 2000

リロケートブルバイナリファイルのアブソリュートバイナリ形式にアセンブルしてローディングを行え
ただし、アセンブルバイアスは1200、ローディングアドレスは2000とする

—— " * L " のコマンド待ちに、L (relocate Load) を入力する。 (" * L " は、 " LOADER " を意味している。)

—— 続けて、アセンブルバイアスとローディングアドレスをそれぞれ16進4桁で入力する。

前項のバイアスの考え方で説明したように、アセンブルバイアスは、プログラムのアブソリュートバイナリ形式を定めるものである。アセンブラで、REL コマンドを実行している場合としていない場合では、同じアブソリュートバイナリを得るのに、異なった指定が必要となる。

一方ローディングアドレスは、一時的にアブソリュートバイナリプログラムをローディングするエリアを決めるものであるから、シンボルテーブルまでのリンクエリア内で、ある程度任意に指定することができる。

—— 次に、システムは " FILENAME? " と表示し、読み込むファイルネームの指定を待つ。

—— ファイルネームを指定したら **[CR]** を押す。システムは PLAY ボタンを押すよう指示する。

—— PLAY ボタンを押すと、指定されたファイルを見つけ出し、読み込みを始める。

ファイルネームを指定していないと、最初に見つけ出したファイルを読み込む。

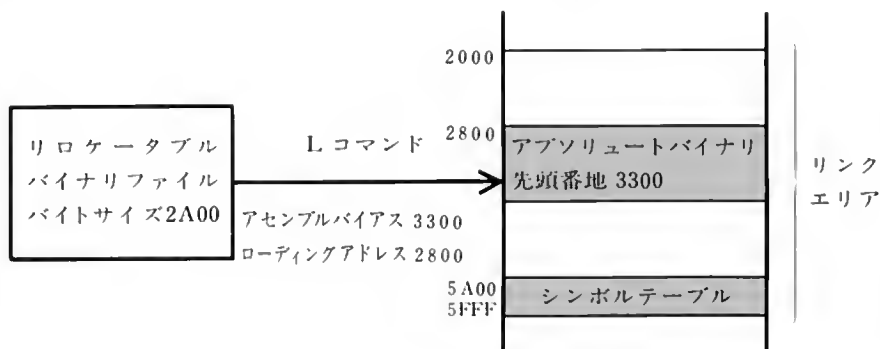
—— 読み込みが終了すると、 " OK " が表示される。

—— 途中でエラーが生じた場合、 " ERROR " を表示する。

—— 読み込みを中止する時は、 **[SHIFT]** **[BREAK]** を押す。

〔問〕 リンクエリアが2000—5FFFであるとする。今、シンボルテーブルを5A00に設定し、バイトサイズ2A00、REL 擬似命令を使っていないリロケートブルバイナリファイルをローディングして、プログラムの先頭番地が3300であるようなアブソリュートバイナリを作成したい時、各バイアスおよびアドレスの設定はいくらとすればよいか。

答……アセンブルバイアスは3300、ローディングアドレスは、2000から3000の範囲で選択できる。たとえば2800（右図）。



N (Next file) コマンド

現在のアセンブルバイアスとローディングアドレスの値（コマンドHで示される値）に従って、次にロードするリロケートブルバイナリファイルをリンキングロードする。

＊ LN リロケートブルバイナリファイルを付加してリンキングロードせよ
アセンブルバイアスとローディングアドレスは現在の値とする

- "＊L" のコマンド待ちに、N (Next file) コマンドを与える。
- システムは "FILENAME?" と表示し、読み込むファイルネームの指定を待つ。
- ファイルネームを指定したら **CR** を押す。システムは PLAY ボタンを押すよう指示する。
- PLAY ボタンを押すと、指定されたファイルを見つけ出し、読み込みを始める。
ファイルネームを指定していないと、最初に見つけ出したリロケートブルバイナリファイルを読み込む。
- ローディングの際のアセンブルバイアスとローディングアドレスは、N コマンドの実行直前の値に従う。この場合、各プログラムユニットで REL コマンドを用いる場合は特に注意が必要であり、通常各ユニットの冒頭に置かれた REL nn' は、すべて nn' の値が一致していなくてはならない。（バイアスの考え方を参照）
- ローディング作業は、プログラムの付加（Append）および、リンク（Linking）が行われる。
- 読み込みが終了すると、"OK" が表示される。
- 途中でエラーが生じた場合、"ERROR" を表示する。
- 読み込みを中止する時は、**SHIFT** **BREAK** を押す。

H (Height) コマンド

＊ LH 現在のアセンブルバイアス、ローディングアドレスの値を表示せよ
（変更は不可）

- "＊L" のコマンド待ちに、H (Height) コマンドを与える。
- システムは、現在のアセンブルバイアスとローディングアドレスの示す値を、それぞれ16進4桁で表示する。
これらのバイアス値を変更することはできない。（＊コマンドで、クリアすることはできる）

〔問〕 L コマンドでアセンブルバイアス1200、ローディングアドレス2000として、バイトサイズ2A00のリロケートブルバイナリファイルをローディングした後、N コマンドでバイトサイズF00のリロケートブルバイナリファイルをリンキングロードした。この時、H コマンドの示す値はいくらか。

答……………アセンブルバイアス 4B00、ローディングアドレス 5900

T (Table dump) コマンド

シンボルテーブルに登録されている内容を表示する。ラベルシンボル名とその絶対アドレス、および定義状態が示される。

* LT シンボルテーブルの内容を表示せよ

- " * L " のコマンド待ちに、 T (Table dump) コマンドを与える。
- システムは、シンボルテーブルに登録されている内容、すなわち、ラベルシンボル名、その絶対アドレス (16 進表現) および、定義状態を表示する。
定義状態を調べることで、シンボル定義の誤りなどを見つけることができる。

—— 右の写真は、 L コマンドを実行した後に、 T コマンドを実行してシンボル定義をチェックするもようを
示している。未定義シンボルには " U " 表示がな
されている。

- シンボルの定義状態に関するメッセージは下表のよ
うになっている。
具体的な例を次頁に示している。



メッセージ	定 義 状 態
U	未定義シンボル Undefined (Adr. or Data)
M	シンボル 2 重定義 Multi-defined (Adr. or Data)
X	クロス定義 Cross-defined (Adr. and Data)
H	データ未定義を含むデータ定義済 Half-defined (Data)
D	データ定義シンボル Data (Data)

アドレス定義済のシンボルにはメッセージがつかない。

* (CLEAR bias and table) コマンド

* L * アセンブルバイアス、ローディングアドレスを0000とし、シンボルテーブルの内容は
クリアせよ

- " * L " のコマンド待ちに、 * (CLEAR) コマンドを与える。
- * コマンドが実行されて、アセンブルバイアス、ローディングアドレスはいずれも0000となり、シンボルテ
ーブルの内容は、クリアされる。
ただし、 " * TBL " でシンボルテーブルに設定された番地はクリアされない。

—各リンクメッセージの説明—

1 番目にロードするプログラムユニット

```
TMDLYH:  LD      HL, START
COUNT:  ENT
          DEC     HL
          LD      A, H
          CP      COUNT0
          JR      NZ, COUNT
          LD      A, L
          CP      COUNT1
          JR      NZ, COUNT
          RET
PEND:    ENT
          DEFM    TMDLYH'
          DEFB    0DH
COUNT1: EQU     00H
COUNT0: EQU     50H

          END
```

"START" H

STARTは1番目のプログラムではData未定義であり、2番目以後のプログラムでSTART: EQUでDataとして定義される。

注

EQU命令はリンクするプログラムユニットの先頭に持ってくる必要がある。

"COUNT1" D

COUNT1がDataとして定義されている。(これはエラーではない)

2 番目にロードするプログラムユニット

```
TMDLYL:  LD      HL, START
LOOP1:   DEC     H
          LD      A, H
          CP      COUNT
          JR      NZ, LOOP
          RET
PEND:    ENT
          DEFM    'TMDLYL'
          DEFB    0DH
START:   EQU     1000H
COUNT:  EQU     00H

          END
```

"COUNT" X

COUNTが1番目にロードしたプログラムでAdr.として定義され2番目にロードしたプログラムでDataとして定義されている。

"PEND" M

PENDが1番目にロードしたプログラムでAdr.として定義され2番目にロードしたプログラムでまたAdr.として定義されている。(二重に定義されている)

3 番目にロードするプログラムユニット

```
INPUT:   CALL    001BH
          CALL    TMDLYL
          CALL    001BH
          LD      HL, START
          CP      0DH
          JR      Z, END
          LD      (HL), A
          INC     HL
          JR      INPUT
END:      JP      0000H

          END
```

"TMDLYL" U

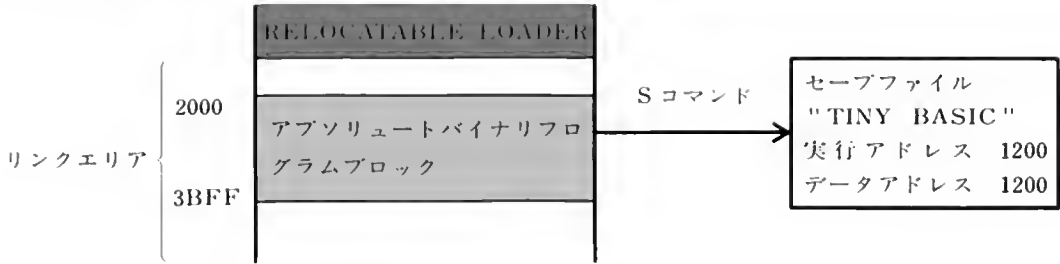
TMDLYLがAdr.として未定義であり、外部プログラムユニットの中でもENT宣言されていない。

S (Save) コマンド

RELOCATABLE LOADER によってリンクエリア内に構成されたアブソリュートバイナリプログラムを、ファイルネームおよび、実行アドレス、データアドレスを指定して、出力ファイルに出力する。この出力によって、目的とするアブソリュートバイナリファイルを得る。

*LS	リンクエリア内の2000から3BFFに構成されている、アブソリュートバイナリブロックを出力せよ
FILENAME? TINY BASIC CR	ただし、ファイルネームは "TINY BASIC" とし、実行アドレス1200、データアドレス1200とする
FROM? 2000 TO? 3BFF	
EXECUTE? 1200 DATA? 1200	

- " *L " のコマンド待ちに、S (Save) コマンドを与える。
 - システムは改行して " FILENAME? " と表示し、出力ファイル名の指定を待つ。
 - ファイルネームを指定したら CR を押す。
 - 改行して " FROM? " を表示し、出力すべきリンクエリア内のアブソリュートバイナリブロックの先頭アドレスが、16進4桁で指定されるのを待つ。
 - 次に " TO? " を表示し、同じくエンドアドレスの16進4桁指定を待つ。
 - 出力するメモリブロックが指定されると、改行して、出力ファイルのインフォメーションブロックに登録する実行アドレス (EXECUTE ADR.) と、データアドレス (DATA ADR.) の指定を持つ。
- 前項 " ロータのバイアスとアドレスの考え方 " で説明したように、実行アドレスは、セーブファイルをロードした後に、プログラムのコントロールが移るアドレス (従ってプログラムカウンタのとり値) であり、データアドレスは、ローディングの行なわれる先頭アドレスになる。
- 以上の指定を終えると、RECORDボタンとPLAYボタンを押すように指示される。
 - RECORDおよびPLAY ボタンを押すと、セーブファイルの出力をはじめる。



- セーブファイルの出力を終了すると " OK " が表示される。
- 途中でエラーが生じた場合、 " ERROR " を表示する。
- ファイル出力を中止する時は、 SHIFT BREAK を押す。

V(Verify)コマンド

ファイルネームで指定されるセーブファイルと、リンクエリアの内容とを比較する。

*LV セーブファイルの内容と、リンクエリアの内容とを比較せよ

- " *L " のコマンド待ちに、V (Verify) コマンドを与える。
 - システムは " FILENAME? " と表示し、比較すべき、セーブファイル名の指定を待つ。
 - ファイルネームを指定したら **CR** を押す。システムは、PLAYボタンを押すよう指示する。
 - PLAYボタンを押すと、指定されたファイルを見つけ出し、比較を始める。
ファイルネームを指定していないと、最初に見つけ出したファイルの比較を行う。
 - 比較されるファイルと、リンクエリアの内容が同一であれば、" OK " と表示され、相違があれば、" ERROR " の表示がなされる。
 - ファイルの比較を中止する時は、**SHIFT BREAK** を押す。
- 右の写真は、セーブファイル " TINY BASIC " とリンクエリアの内容とを比較し、互いに一致したものを示している。

```
#LV  
FILENAME?TINY BASIC  
↓ PLAY  
FOUND TINY BASIC  
VERIFYING TINY BASIC  
OK  
#L
```

〔問〕 リンクエリア内の2000から2F00までに、5E00を先頭番地とする形式のアブソリュートバイナリプログラムがある。BASICの機械語サブルーチン群であるこのアブソリュートバイナリプログラムを、ファイルネーム " SUB-ROUTINE " なるセーブファイルに出力する方法を示せ。

答……右の写真に示す。

このセーブファイルは、モニタでロードされて単独で用いられるものではなく、BASICプログラムのサブルーチンである。すなわち、BASICのLIMITコマンドによって確保された領域に、LOADコマンドによってリンクされる。そうした場合（モニタによるロード以外の場合）実行バイアスは無視され、ロードを行なったシステムプログラムにコントロールは戻る。右の例では実行バイアスは便宜的に0000としてある。（セーブファイルとBASICとのリンク方法は、P.87を参照）

```
#LS  
FILENAME?SUB-ROUTINE  
FROM? 2000 TO? 2F00、  
EXECUTE? 0000 DATA? 5E00  
↓ RECORD.PLAY
```

コマンド

* L # プリンタへのリスティングのためのリストモードを切り替えよ

—— " * L " のコマンド待ちに、 # (sharp mark) コマンドを与える。

—— プリンタへのリスティングのための、リストモードが切り替わる。

RELOCATABLE LOADER の起動時は、プリンタリストモードは disable であり、 # コマンドを 1 回与えるごとに、モードは enable、disable と切り替わる。

プリンタリストモードが enable であると、出力表示は、テレビ画面と、プリンタの両方に行われる。

—— 下図は、オプションの放電プリンタへ、T コマンドによるシンボルテーブルのリストを行った印字例である。

*LT							
?PRNT	5380	BELL	D 000E	BRKEY	D 001E	BSIZE0	5380
BUFF0	5400	BUFF1	540A	BUFF2	540C	BUFF3	540E
BUFF4	5410	COMMN	524A	DISPL	52DA	DSPXY	D 1171
FREE	5446	GETKY	D 001B	GETL	D 0003	HEIG	5205
KANAF	D 1170	KANST	D E000	KEYPA	D E000	KEYPC	D E002
LETNL	D 0009	LOOK1	527D	LOOK2	5293	MAIN0	5200
MAIN1	5217	MAIN2	5256	MELDY	D 0030	MSG	D 0015
MSG0	5412	MSG1	5417	MSG2	541B	MSG3	5421
MSTP	D 0047	NL	D 0006	PRNT	D 0012	PRNTS	D 000C
READ	52A4	READ0	52B0	RLINK	5229	RSAVE0	525E
RSAVE1	5279	SEARCH	52E4	SORT	5473	START	5200
STCK	53B5	STIN	53EC	TAPE	5043	TEMP0	5420
TEMP1	542D	TEMP2	5435	TEMP3	543B	WARN	5366
WRITE	52CA	XTEMP	D 0041				

! コマンド

* L ! モニタへコントロールを移せ

—— " * L " のコマンド待ちに、 ! (exclamation mark) コマンドを与える。

—— システムのコントロールは、モニタへ移る。

—— モニタから、RELOCATABLE LOADER へ戻るには、次の 2 通りの方法がある。

* GOTO \$ 1200 CR リンクエリアの内容をクリアして、スタックを初期状態に戻す。
(COLD START)

* GOTO \$ 1260 CR リンクエリアの内容はクリアせず、コマンド待ちへ移る。
(WARM START)

第 5 章

SYMBOLIC DEBUGGER

SP-2401

mz-80k



5-1 シンボリックデバッグ概要

シンボリックデバッグ SP-2401 は、アセンブラ出力のリロケートブルバイナリファイルをリンクしながら即時実行可能な、絶対型式オブジェクトプログラムをメモリ上に構成し、プログラムを実際に走らせてデバッグ操作を行うことができる。

デバッグ操作は、プログラム中に**ブレイクポイント**を設定して、プログラムの実行をそこで中止させることによって行う。ブレイクポイントがかかった時、レジスタの内容は待避される一方、その内容を変更したり、バッファとして使用しているメモリエリアの内容を調べることができる。チェック、変更を行った後、レジスタの内容を復帰してその箇所からのリスタートが可能である。

ブレイクポイントの設定や、メモリダンプ、実行コマンドなどは、アドレス指定の際必ずしも絶対アドレスで指定する必要はなく、ソースプログラム中で、エントリ宣言 (擬似命令 ENT) のなされたラベルシンボルを用いて相対的に指定することができる。

デバッグが終了し、プログラム中の誤りを見つけ出したならば、ソースプログラムを編集し直す。各ソースプログラムユニットのデバッグが全て終了したら、リロケートブルローダを用いて目的とするオブジェクトバイナリファイルを得ることになる。

シンボリックデバッグのコマンドは次のものがある。このうち+マークを付したコマンドは、シンボリック操作が可能なものである。尚、シンボルテーブルの設定はリロケートブルローダの場合と同様である。

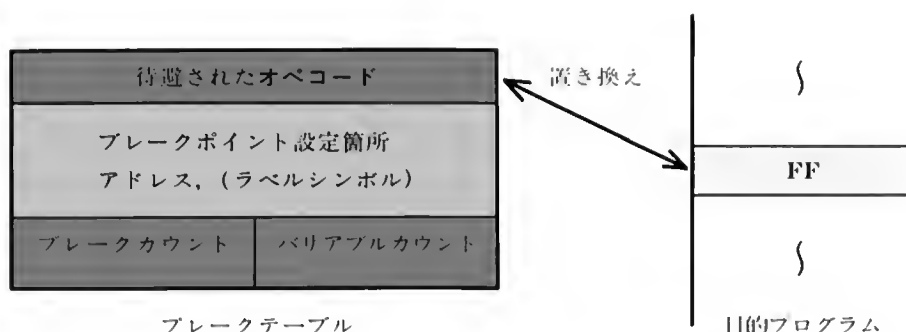
コマンドの種類	コマンド名	機 能
リンキングロード シンボルテーブルコマンド	L	リンクエリア内に、アセンブラ出力のリロケートブルバイナリファイルを入力する。リロケートブルバイナリは、アセンブルバイアス、ローディングアドレスで指定された形式のアブソリュートプログラムとしてストアされる。(relocate Load)
	N	リンクエリア内のプログラムに、入力リロケートブルバイナリファイルを、付加およびリンクしてローディングする。(Next file)
	H	現在のアセンブルバイアス、ローディングアドレスの値を表示する。(Height)
	T	シンボルテーブルに登録されている内容を表示する。ラベルシンボル名とその絶対アドレスおよび定義状態が示される。(Table dump)
	*	現在のアセンブルバイアス、ローディングアドレス値をクリアして0000とする。またシンボルテーブルの内容をクリアする。(CLEAR bias and table)
デバッグ コマンド	B†	ブレイクポイントを設定する。(Break point)
	&	設定されたブレイクポイントをすべてクリアする。(clear break point)
	M†	リンクエリア内の指定したブロックの内容を16進表示する。(Memory dump)
	D†	リンクエリア内の指定したブロックの内容を命令単位で16進表示する。(memory list Dump)
	W†	リンクエリア内の指定したアドレスから、データを16進コードで書き込む。(Write)
	G†	プログラムを指定アドレスから実行する。(Goto)
	I	レジスタバッファの示す内容でプログラムを実行する。実行アドレスはPCの示す値である。(Indicative start)
	A	A, F, B, C, D, E, H, L の内容を16進表示、または変更する。(Accumulator)
	C	A', F', B', C', D', E', H', L' の内容を16進表示、または変更する。(Complementary)
	P	PC, SP, IX, IY, I の内容を16進表示、または変更する。(Program counter)
	R	A, C, P コマンドのすべてのレジスタをまとめて16進表示する。(Register)
	X	指定されたメモリブロックを指定アドレスへ転送する。(TRANS fer)
アブソリュート ファイル入 出力コマンド	S	リンクエリア内のアブソリュートプログラムとシンボルテーブルとを出力ファイルへファイルネームを指定して出力する。(Save)
	V	ファイルネームで指定されたファイルとメモリの内容とを比較する。(Verify)
	Y	ファイルネームで指定されたアブソリュートファイルをメモリに読み込む。(Yank)
特殊コマンド	#	プリンタへのリスティングのため、リストモードを切り替える。
	!	モニタへコントロールを移す。

5-2 ブレークポイントの考え方

ブレークポイント (break point) とは、プログラム中に設定するチェックポイントのことであり、ブレークポイントとして指定された箇所でプログラムの実行を停止し、CPUレジスタの内容を、レジスタバッファに待避させる。その状態でプログラマは、レジスタの内容やメモリの内容を調べたり、修正を加えたりすることができ、そのあとで、プログラムのリスタートを行なうこともできるので、プログラムのチェック、およびデバッグが可能である。

シンボリックデバッガで設定できるブレークポイントは、最大9箇所までとなっている。ブレークポイントの設定は、位置（アドレス）を指定するだけでなく、カウントも指定しなくてはならない。カウントというのは、ループするプログラムで、カウントとして指定した回数プログラムの実行がその箇所へ来たとき、はじめてブレーク動作がかかるようにするために指定するものである。ブレークカウンタの最大値はE（14回）である。

ある箇所にブレークポイントを設定すると、その位置（アドレス）にあるオペコードを、ブレークテーブルへ待避させ、そのかわりにFFコードで置き換える。1つのブレークポイントについて、次に示すブレークテーブルが作成される。



16進FFコードは、RST 7のオペコードであり、これがブレーク動作を行なうことになる。1バイトのCALL命令であるRST 7命令を実行すると、はじめプログラムカウンタの内容がブッシュされ、新たに0038Hがプログラムカウンタの内容となる。即ち、モニタ内の0038番地へジャンプするが、そこからデバッガへコントロールが移って来る。そしてブレークテーブルを検索し該当するブレークポイントを見つけ出す。(見つからなかった場合は、エラー表示として"RST 7?"が表示される。本システムではRST 7はモニタ内で特殊な実行が行なわれるので通常、使用することはできない。)

ブレークポイントをテーブル内に見つけたら、カウントを調べる。最初、ブレークカウント=バリエابلカウントであったものを、バリエابلカウントをデクリメントし、それが0になったら、ブレーク動作を行うし、そうでなければ、プログラムを継続させることになる。

5-3 シンボリックデバッグコマンド

——リンクグロードコマンド——

L (relocate Load) コマンド

リンクエリア内にアセンブラ出力のリロケートブルバイナリファイルを入力する。デバッガは、プログラムを実際に走らせることを前提としているので、このコマンドで指定すべきバイアス値は、それを考えたものでなければならないことになる。

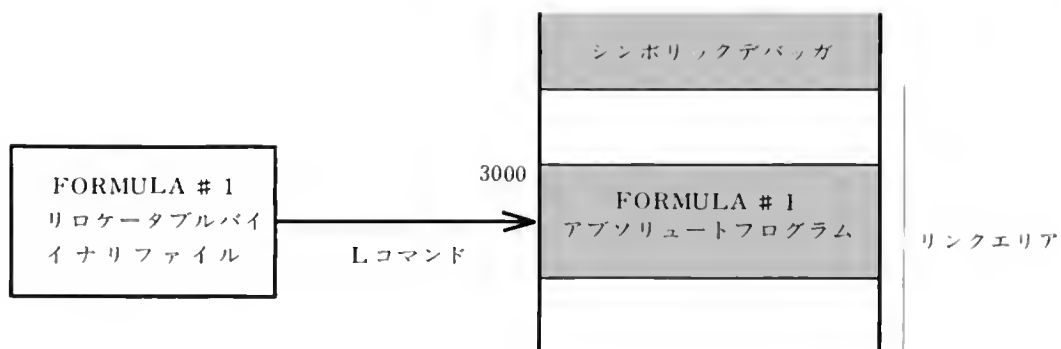
* DL 3000 3000

リロケートブルバイナリファイルをロードせよ

ただし、アセンブルバイアスは3000、ローディングアドレスは3000とする

- " *D " のコマンド待ちに、 L (relocate Load) を入力する。 (" *D " は " Debugger " を意味している。)
- 続けて、アセンブルバイアスとローディングアドレスをそれぞれ16進4桁で入力する。前章のリロケートブルロードで説明されたバイアスの考え方に従って、リンクエリア内に、即時実行可能なアブソリュートプログラムを構成する。アセンブラで REL nn の操作を実行していない通常の場合は、上の例のように、アセンブルバイアスとローディングアドレスは同一であり、それは、リンクエリア内のあるアドレスに決められるはずである。
- 次に、システムは " FILENAME? " と表示し、読み込むファイルネームの指定を待つ。
- ファイルネームを指定したら [CR] を押す。システムは PLAY ボタンを押すよう指示する。
- PLAY ボタンを押すと、指定されたファイルを見つけ出し、読み込みを始める。
ファイルネームを指定していないと、最初に見つけたファイルを読み込む。
- 読み込みが終了すると、 " OK " が表示される。
- 途中でエラーが生じた場合、 " ERROR " を表示する。
- 読み込みを中止する時は、 [SHIFT] [BREAK] を押す。
- 右の写真は、リロケートブルバイナリファイル " FORMULA # 1 " (REL を用いていない) を、リンクエリア内の3000番地からアブソリュート形式で読み込むようが示されている。

```
## SYMBOLIC DEBUGGER SP-2401 ##
LINK AREA 258F - CFFF
#TBL B0
#DL 3000 3000
FILENAME?FORMULA#1
↓PLAY
FOUND FORMULA#1
LOADING FORMULA#1
```



N (Next file) コマンド

現在のアセンブルバイアスとローディングアドレスの値（コマンドHで示される値）に従って、次に入力するリロケートブルバイナリファイルをリンキングロードする。

*** DN** リロケートブルバイナリファイルを付加してリンキングロードせよ
アセンブルバイアスとローディングアドレスは現在の値とする

- " *D " のコマンド待ちに、N (Next file) コマンドを与える。
- システムは " FILENAME? " と表示し、読み込むファイルネームの指定を待つ。
- ファイルネームを指定したら **[CR]** を押す。システムはPLAYボタンを押すよう指示する。
- PLAYボタンを押すと、指定されたファイルを見つけ出し、読み込みを始める。
ファイルネームを指定していないと、最初に見つけ出したファイルを読み込む。
- ローディングの際のアセンブルバイアスとローディングアドレスは、Nコマンドの実行直前の値に従う。この場合、各プログラムユニットでRELコマンドを用いる場合は特に注意が必要であり、通常各ユニットの冒頭に置かれたREL nn' は、すべてのnn'の値が一致していなくてはならない。(前章を参照)
- ローディング作業では、プログラムの付加 (Append) および、リンク (Link) が行われる。
- 読み込みが終了すると、" OK " が表示される。
- 途中でエラーが生じた場合、" ERROR " を表示する。
- 読み込みを中止する時は、**[SHIFT]** **[BREAK]** を押す。

H (Height) コマンド

*** DH** 現在のアセンブルバイアス、ローディングアドレスの値を表示せよ
(変更は不可)

- " *D " のコマンド待ちに、H (Height) コマンドを与える。
- システムは、現在のアセンブルバイアスとローディングアドレスの示す値を、それぞれ16進4桁で表示する。
これらのバイアスおよびアドレス値を、変更することはできない。

〔問〕 Lコマンドでアセンブルバイアス3000、ローディングアドレス3000として、リロケートブルバイナリファイル " FORMULA # 1 " (バイトサイズ 500H バイト) をロードしたあと、Hコマンドで示されるバイアスおよびアドレス値はいくらであるか。

答……………アセンブルバイアス3500、ローディングアドレス3500である。

〔問〕 今のリロケートブルバイナリファイル " FORMULA # 1 " が冒頭に REL 4000H 命令を使用しており、Lコマンドでアセンブルバイアス0000、ローディングアドレス4000でローディングした場合はどうか。

答……………アセンブルバイアス0500、ローディングアドレス4500となる。アブソリュートプログラムは4000番地から実行可能なものになり、ローディングの先頭も4000番地になる。

T (Table dump) コマンド

シンボルテーブルに登録されている内容を表示する。ラベルシンボル名とその絶対アドレス、および定義状態が示される。

*DT

シンボルテーブルの内容を表示せよ

- "＊D" のコマンド待ちに、T (Table dump) コマンドを与える。
- システムは、シンボルテーブルに登録されている内容、すなわち、ラベルシンボル名、その絶対アドレス (16進表現)、および定義状態を表示する。
定義状態を調べることにより、シンボル定義の誤りなどを見つけることができる。
- シンボルテーブルの定義状態に関するメッセージは、リロケートブルロードの場合と同様に次のようになっている。具体的な例は61ページに説明がある。

メッセージ	定 義 状 態
U	未定義シンボル Undefined (Address or Data)
M	シンボル 2 重定義 Multi-defined (Address or Data)
X	クロス定義 Cross-defined (Address and Data)
H	データ未定義を含むデータ定義済 Half-defined (Data)
D	データ定義シンボル Data (Data)

アドレス定義済のシンボルにはメッセージがつかない。

* (CLEAR bias and table) コマンド

* D *

アセンブルバイアス、ローディングアドレスを0000とし、シンボルテーブルの内容はクリアせよ

- "＊D" のコマンド待ちに、＊ (CLEAR) コマンドを与える。
- ＊コマンドが実行されて、アセンブルバイアス、ローディングアドレスはいずれも0000となり、シンボルテーブルの内容はクリアされる。
ただし、"＊TBL" でシンボルテーブルに設定された番地はクリアされない。

B (Break point) コマンド

ブレークポイントを設定または変更するコマンドである。ブレーク動作は、ブレークポイントの設定されたアドレスの1つ前までの命令をブレークカウンタに指定した回数だけ実行した時に行なわれ、そこでプログラムの実行が中断され、同時にCPUレジスタの内容がレジスタバッファに待避されて、コマンド待ちへ戻る一連の動作からなる。ブレークポイントとして指定するアドレスは、16進絶対アドレスまたはラベルシンボルを用いた指定ができる。

*DB	ブレークポイントを設定する
ADDR COUNT	
1 5030 _2	アドレス5030、ブレークカウント2
2 SORT3 _1	ラベルシンボル "SORT3" のアドレス、ブレークカウント1
3 SORT3+5L _1	ラベルシンボル "SORT3" の5行先のアドレス、ブレークカウント1
4 MAIN0-A _2	ラベルシンボル "MAIN0" のAバイト前のアドレス、ブレークカウント2
5 ☒	(_ はスペース記号、スペースを入力して2つのデータを区別すること)

—— " *D " のコマンド待ちに、B (Break point) コマンドを与える。

—— システムは改行して " ADDR COUNT " と表示する。

さらに改行してブレークポイントナンバを表示し、スペースを1個おいてカーソルを表示してブレークポイントアドレスとブレークカウンタの入力待ちとなる。

ブレークポイントアドレスの指定は、16進4桁または、グローバルシンボルを用いたシンボリックな指定が可能である。(上例) いずれの場合も、アドレス指定後、スペースを1個おいて、ブレークカウンタを指定する。ブレークカウンタは、ブレークポイントの一つ前の命令をその回数実行した時にブレーク動作を行わせるためのものであり、1からE回までの間で指定できる。

ブレークカウンタを指定すると、改行して、次のブレークポイントナンバを表示して、ブレークポイントの入力待ちとなる。

—— ラベルシンボルを用いたアドレス指定を行った時、該当するシンボルが登録されていないか、そのシンボルがデータ定義シンボルである場合 " ??? " を表示してコマンド待ちに戻る。

—— DJNZ 命令には、ブレークポイントは設定できない。もし設定しようとした場合は " DJNZ ? " の表示がなされ、コマンド待ちに戻る。

—— 同じく CALL 命令にもブレークポイントを設定することはできない。プログラムカウンタをスタックする命令にブレークポイントを設定できない。エラー表示は " CALL ? " である。

もし CALL 命令のチェックをしたい場合には、そのコール先にブレークポイントを設定する方法がある。

—— 前に設定したブレークポイントを解除したい時は、同じアドレスを入力し、ブレークカウンタを0とすればよい。(ブレークポイントを全て解除するには、次項に示す & コマンドがある。)

設定されていないブレークポイントを解除しようとする時、 " ??? " を表示しコマンド待ちに戻る。

—— ブレークポイントは、最大9個まで設定できる。9個まで指定すると改行して、ブレークポイントナンバの所に " X " と表示して、同様なコマンド待ちとなるが、これは、ブレークポイントの解除または、ブレークカウンタの変更を行うためのものであり、新しいブレークポイントを設定するためのものではない。 " X " の所に新しいブレークポイントを設定しようすると受け付けられず " OVER " を表示してコマンド待ちに戻る。

—— ブレークポイントを設定した後、再びBコマンドを実行すると、設定されたブレークポイントが表示されるが、この時、設定した時のテレビ画面とは異り、16進アドレス、ブレークカウンタ、設定の時用いたシンボルの順で整理されて表示される。

—— カーソル操作ではDELETE ([DEL] キー) 機能を行うことができる。 [CR] キーを押すとコマンド待ちに戻る。

&(CLEAR B.P)コマンド

*D& 設定されたブレイクポイントをすべて解除せよ

- " *D " のコマンド待ちに、 & (CLEAR break point) コマンドを与える。
- システムは、設定されているブレイクポイントをすべて解除して、次のコマンド待ちとなる。

—— 右の写真は、ブレイクポイントを設定しているもようを示す。16進4桁による絶対アドレス指定、グローバルラベルシンボルを用いたアドレス指定、ラベルシンボルにライン、またはバイト単位のディスプレイメントを加えた場合などが示されている。

```
#DB
ADDR COUNT
1 5050 1
2 MAIN0 2
3 MAIN8+3L 1
4 LIST-1C 2
5 5055 3
6 兼
```

—— 右の写真は、" X " 番目の項で、ブレイクポイント " SORT3 " の解除を行ったもようを示している。

```
#DB
ADDR COUNT
1 5050 1
2 MAIN0 2
3 MAIN8+3L 1
4 LIST-1C 2
5 5055 3
6 SORT3 1
7 SORT3+1L 1
8 SORT3+2L 1
9 5080 3
X SORT3 0

#兼
```

—— 右の写真は、ブレイクポイントの設定後、Bコマンドでブレイクポイントを表示させたものである。各ブレイクポイントは、16進絶対アドレス、ブレイクカウント、そして設定の際にラベルシンボルを用いた場合には、そのシンボルの順に整理されて表示されている。

```
#DB
ADDR COUNT
1 5050 1
2 5065 2
3 506F 1 READ0
4 50A2 3 KOELL+3L
5 兼
```

—— 右の写真は、ブレイクポイントを5000、カウント1として、Gコマンドで5000からプログラムを実行させ、即座にブレイク動作を行わせたもようを示す。ブレイク動作が行われると同時に、Rコマンドが実行されて、レジスタバッファ内に待避されたCPUレジスタの値を見ることができる。

```
#DB
ADDR COUNT
1 5000 1
2

#DG 5000
A F B C D E H L
01 23 45 67 89 AB CD EF
A' F' B' C' D' E' H' L'
01 23 45 67 89 AB CD EF
PC SP IX IY I
5000 11F3 0000 0000 00

#D兼
```

M (Memory dump) コマンド

指定されたメモリブロック内の機械語データを16進表示する。メモリブロックの指定は16進絶対アドレスまたはラベルシンボルを用いた指定ができる。カーソル操作によってデータの書き換えも可能である。

*DM 3300┐3350 **[CR]**

3300から3350までのメモリブロックの内容を表示せよ

*DM MAIN7┐MAIN9 **[CR]**

ラベルシンボル"MAIN7"の示すアドレスからラベルシンボル"MAIN9"の示すアドレスまでのメモリブロックの内容を表示せよ

*DM STEP0-A┐STEP3+BL **[CR]**

ラベルシンボル"STEP0"の示すアドレスのAバイト前から、ラベルシンボル"STEP3"のBライン先のアドレスまでのメモリブロックの内容を表示せよ

—— " *D " のコマンド待ちに、 M (Memory dump) コマンドを与える。

—— システムは、スペースを1個おいてカーソルを表示し、メモリダンプを行うメモリブロックの先頭アドレスとエンドアドレスの入力待ちとなる。

メモリブロックのアドレス指定は、16進4桁または、グローバルシンボルを用いたシンボリックな指定が可能である。

—— メモリブロックのアドレス指定は、先頭アドレス ≤ エンドアドレス でなくてはならない。そうでない場合は " ? " の表示がなされる。

—— 表示できるメモリブロックは、リンクエリアでなければならない。

*DM 1100 1150

1100

???

} 1100~1150はリンクエリアではない。

—— リンクエリア内のメモリブロックが指定されると、1行8バイトずつテレビ画面へメモリダンプが行われる。

—— プリンタ出力モードが enable の場合、プリンタ上へ表示されるメモリダンプリストは、1行16バイトずつ行われる。

—— テレビ画面への表示では、メモリブロックのダンプの後にカーソルが現われ、カーソル操作によって、データの書き換えを行うことができる。変更は、書き換えのあと **[CR]** キーを押すことによって実行される。

データの書き換えは表示されているデータの上から行う。書式が合わないとき " ERROR " 表示がなされる。

—— **[CR]** はカーソルのある行の入力を実行するが、もしデータのない行の先頭にある場合に **[CR]** キーを押すと、次のコマンド待ちとなる。

—— 強制的にコマンド待ちに戻すには **[SHIFT] [BREAK]** キーを押す。

D(memory list Dump)コマンド

指定されたメモリブロック内の機械語データを、命令単位で1行ごとに並べたダンプリストとして16進表示する。メモリブロックの指定は16進絶対アドレスまたはラベルシンボルを用いた指定ができる。カーソル操作によるデータの書き換えはできない。

*DD 3300 _ 3350 **CR**

3300から3350までのメモリブロックの内容を命令単位で表示せよ

*DD START _ MAIN0 **CR**

ラベルシンボル "START" の示すアドレスから、ラベルシンボル "MAIN0" の示すアドレスまでのメモリブロックの内容を命令単位で表示せよ

*DD 3000 _ START+12L **CR**

3000番地から、ラベルシンボル "START" の12行先のアドレスまでのメモリブロックの内容を命令単位で表示せよ

— " *D " のコマンド待ちに、D (memory list Dump) コマンドを与える。

— システムはスペースを1個おいてカーソルを表示し、メモリリストダンプを行うメモリブロックの先頭アドレスとエンドアドレスの入力待ちとなる。

メモリブロックのアドレス指定は、16進4桁または、グローバルシンボルを用いたシンボリックな指定が可能である。メモリブロックのアドレス指定はMコマンドの場合と同様に、先頭アドレス ≤ エンドアドレスで、かつ、リンクエリア内のメモリブロックでなければならない。

メモリブロックの指定後 **CR** キーを押すと、システムは、アドレスと命令単位の機械語コードとを並べて順次表示を行う。

例えば、"START" から "MAIN0" までが次のようなソースプログラムによって構成されており、デバッガで3000番地からストアされているとすると、Dコマンドでリストダンプを行うと右の写真のように表示が行われる。

```
START: ENT
        LD    SP, START
        CALL MSTP
        XOR   A
        LD    (?TABP), A
        LD    B, A
MAIN0: ENT
        LD    A, OFH
```



```
#DD START MAIN0
3000 310030
3003 CD4700
3006 AF
3007 32BE30
300A 47
300B 3E0F
```

#D

— 注意しなくてはならないのは、Dコマンドで指定するメモリブロックの先頭アドレスは必ずある命令のオペコードを指していなければならない点である。そうでなくデータを指していたりするとそれをオペコードと読んだ不可解なダンプリストが表示される。メモリブロック内にデータ領域 (DEFB, DEFW, などで構成したものなど) がある場合も同様である。

— メモリリストダンプは、**SPACE** キーによって停止、続行を行うことができる。

— メモリ内容の書き換えはできず、メモリブロックの表示のあと次のコマンド待ちとなる。

— コマンドを中断するには **SHIFT** **BREAK** を押す。

W(data Write)コマンド

指定されたメモリアドレスから、16進データを書き込む。メモリアドレスの指定は16進絶対アドレス、またはラベルシンボルを用いた指定ができる。

*DW 8000 **CR**

8000番地から機械語データを書き込む

*DW DATA1 **CR**

ラベルシンボル "DATA1" の示すアドレスから機械語データを書き込む

- " *D " のコマンド待ちに、W (data Write) コマンドを与える。
- システムはスペースを 1 個おいてカーソルを表示し、データライトを行うメモリエリアの先頭アドレスの入力待ちとなる。
メモリエリアの先頭アドレスの指定は、16進 4 桁または、グローバルシンボルを用いたシンボリックな指定が可能である。
- 書き込みを行うメモリエリアはリンクエリア内ではない。



*DW 1111

1111

?

1111番地はリンクエリア内ではない

- アドレスを指定して **CR** キーを押すと改行してアドレスを表示し、先頭アドレスから順次機械語データが16進 2 桁で入力されるのを待つ。
データは 2 桁入力される毎に、スペースが自動的に 1 個置かれる。さらに 8 バイトデータが書き込まれる度に改行して新しくアドレス表示がなされる。

- タイプミスの修正を行うには、 のカーソル左移動キーを押すことによって 1 バイト前へ戻り訂正を行う。右の写真はそのもようを示しており、 キーを押すと改行して戻ったアドレスを表示することがわかる。

- Z80の相対ジャンプ命令JR、DJNZ命令などのディスプレイメントeを指定する場合、ピリオド"." を入力するとシステムは、16進 4 桁のジャンプ先(ラベルは不可)の絶対アドレス入力待ちとなる。16進 4 桁のアドレスを直接指定すると、システムは自動的にディスプレイメントeを算出して所定のアドレスへ 1 バイトのコードをストアする。
右の写真の下の部分はこのように示している。



- 必要なデータの書き込みが終わったら **CR** キーを押す。システムはコマンド待ちに戻る。

G (Goto) コマンド

実行コマンドであり、プログラムのコントロールを指定されたアドレスへ移す。ブレーク動作が行われたあとのリスタートでも用いる。

*DG 3200 [CR]	3200番地からプログラムを実行せよ
*DG START [CR]	ラベルシンボル "START" の示すアドレスからプログラムを実行せよ
*DG [CR]	ブレークポイントからのリスタートを行え リスタート番地、CPUレジスタの値は、レジスタバッファの示す内容とせよ

—— *D " のコマンド待ちに、G (Goto) コマンドを与える。

—— システムは、実行アドレスの入力待ちとなる。実行アドレスは、16進4桁または、ENT擬似命令で定義されたグローバルラベルシンボルで指定することができる。

ラベルシンボルを用いたアドレス指定では、ライン単位のロケーション、またはバイト単位のロケーションを用いることができる。

*DG MAIN0	実行アドレスを "MAIN0" とせよ。
*DG MAIN0+3L	実行アドレスを "MAIN0" より3ライン先の番地とせよ。
*DG MAIN0-B	実行アドレスを "MAIN0" よりBバイト前の番地とせよ。

—— ブレークポイントからのリスタートを行う時は、Gコマンドを与えたあと、[CR] キーを押す。ブレーク動作を行っていないのにこの操作を行うと、実行はせずコマンド待ちに戻る。

リスタートの時に、各CPUレジスタの値は、Rコマンドで示される値である。PC (Program Counter) の値がリスタートの番地となる。PコマンドでPCの値を変更することができるので、リスタートをブレークポイント以外の場所から行うことも可能である。

—— プログラムを実行させ、ある箇所からデバッグへ戻るには、次のコマンドを置いておけばよい。

JP 1260H


1260番地は、デバッグのWARM START番地であり、リンクエリアの内容を消去せずに、" *D " のコマンド待ちとなる番地である。(1200番地からのスタートはリンクエリア、シンボルテーブル、バイアスをクリアするCOLD STARTである)

—— プログラムの実行を停止するには、デバッグ (或はモニタ) へのジャンプ命令を置くか、ブレークポイントによるしかない。

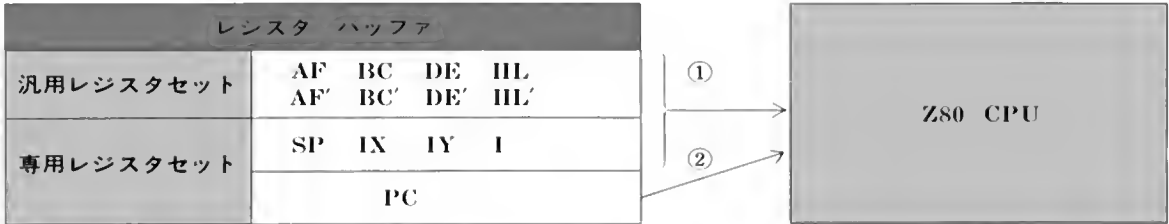
—— Gコマンドの入力を中止する場合は [SHIFT] [BREAK] を押す。

I (Indicative start)

レジスタバッファの示す値をCPUレジスタの値としてプログラムの実行を行うコマンドである。実行アドレスは、PCの示す値であり、これらのレジスタに設定される値は、A、C、Pコマンドでカーソル操作を用いて任意に定めることができる。Gコマンドは単に与えられたアドレスからスタートするのみであり、各レジスタのセットは行なっていない。

*DI								CPU レジスタのとり値を次の値として PCの示すアドレスからプログラムを実 行せよ
A	F	B	C	D	E	H	L	
01	23	45	67	89	AB	CD	EF	
A'	F'	B'	C'	D'	E'	H'	L'	
01	23	45	67	89	AB	CD	EF	
PC	SP		IX		IY		I	
33AB	1FEA		5F70		5F50		00	
START	OK? 							

- " *D " のコマンド待ちに、I (Indicative start) コマンドを与える。
- システムは、スタートの条件となるCPUレジスタに設定する各レジスタの値を16進 2 桁または 4 桁で表示する。これらはレジスタバッファの内容であり、Rコマンドで見ることのできる値と同一内容である。
- レジスタバッファの内容の表示につづいて、" START OK? " の表示が行われ、この条件でスタートする場合は **CR** キーを押す。システムは、PCの示すアドレスから以上の条件でプログラムの実行を行う。レジスタの値を変更するか、Iコマンドを中止する場合は **SHIFT** **BREAK** を押すとコマンド待ちに戻る。
- 下図は、IコマンドによるCPUレジスタ設定のようを示す。



はじめに汎用レジスタと専用レジスタSP、IX、IY、Iの値がCPUレジスタに設定され①、次にプログラムカウンタPCが設定されて②、プログラムが実行される。

右の写真は、所定のレジスタ値を設定し、3000番地からプログラムの実行を行うようを示している。



A (Accumulator) コマンド

ブレークポイントでブレークがかかった場合、Z80 CPUの内部レジスタの内容はバッファに待避されるが、A コマンドによって、そのうちの汎用レジスタ中の主レジスタセットの内容を表示させることができる。更にカーソル操作でそのバッファの内容を変更させ、ブレークポイントからのリスタートが可能である。

*DA	主レジスタAF, BC, DE, HLの内容を 表示せよ							
A	F	B	C	D	E	H	L	
01	23	45	67	89	AB	CD	EF	
❖								

- " *D " のコマンド待ちに、A (Accumulator) コマンドを与える。
- システムは、アキュムレータA、フラグレジスタF、汎用ペアレジスタBC, DE, HLのそれぞれの内容を16進2桁で表示する。
ここでレジスタの内容とは、ブレークポイントがかかった時の各レジスタの内容であり、ブレークポイントからのリスタート (G コマンド参照) のために待避されたものである。
- レジスタの内容が表示された次の行にカーソルが現われる。必要があれば、レジスタの内容を変更する。値の変更は、表示されている値の上にカーソルを持って行き同じ位置に新しい値を書き込む。変更したら CR キーを押す。(カーソルは次の行へ移る)
Aコマンドで表示されたレジスタの内容、あるいは変更された値はブレークポイントからのリスタート、Iコマンドによるインディカティブスタートの場合にCPU内部レジスタに復帰する値である。
カーソルが上の例に示す位置にあるとき、CR キーを押すと、次のコマンド待ちに戻る。

C (Complementary) コマンド

ブレーク動作の行われた時のCPU汎用レジスタ群のうちの、補助レジスタセットの内容を表示する。カーソル操作によって内容の変更を行うことができる。

*DC	補助レジスタAF', BC', DE', HL', の 内容を表示せよ							
A'	F'	B'	C'	D'	E'	H'	L'	
01	23	45	67	89	AB	CD	EF	
❖								

- " *D " のコマンド待ちに、C (Complementary) コマンドを与える。
- システムは補助レジスタのアキュムレータA'、フラグレジスタF'、汎用ペアレジスタBC', DE', HL'のそれぞれの内容を16進2桁で表示する。
各レジスタの内容、または、カーソル操作で変更された値の持つ意味は、Aコマンドの場合と同様であり、ブレークポイント、ブレークポイントからのリスタート、あるいはIコマンドの場合にのみ有効である。
- カーソルが上の例に示す位置にあるとき、CR キーを押すと、次のコマンド待ちに戻る。

P (Program counter) コマンド

ブレイク動作の行われた時のCPU専用レジスタセットの内容を表示する。カーソル操作によって内容の変更を行うことができる。

*DP					専用レジスタ PC, SP, IX, IY, I の内容
PC	SP	IX	IY	I	を表示せよ
33AB	1FEA	5F70	5F50	00	
✖					

- " *D " のコマンド待ちに、 P (Program counter) コマンドを与える。
- システムは、専用レジスタ PC, SP, IX, IY, I のそれぞれの内容を16進 4 桁、または 2 桁で表示する。
各レジスタの内容、または、カーソル操作で変更された値の持つ意味は、A コマンド、C コマンドの場合と同様である。
表示された値、あるいはカーソル操作によって変更された値はブレイクポイントからのリスタート、あるいは I コマンドによるインディカティブスタートの場合にレジスタに復帰される値である。ブレイクポイントからのリスタートは、必ずしもブレイクポイントの置かれたアドレスからでなくてよく、 PC の値を変更することによって、リスタートするアドレスを任意に決めることができる。
- カーソルが上の例に示す位置にあるとき、 CR キーを押すと、次のコマンド待ちに戻る。

R (Register)

ブレイク動作の行われた時のCPUレジスタの内容、あるいは、A、C、Pの各コマンドによって変更された値を表示する。即ち、レジスタバッファの内容が表示されるが変更することはできない。

*DR								CPU レジスタの内容を表示せよ
A	F	B	C	D	E	H	L	
01	23	45	67	89	AB	CD	EF	
A'	F'	B'	C'	D'	E'	H'	L'	
01	23	45	67	89	AB	CD	EF	
PC	SP	IX	IY	I				
33AB	1FEA	5F70	5F50	00				

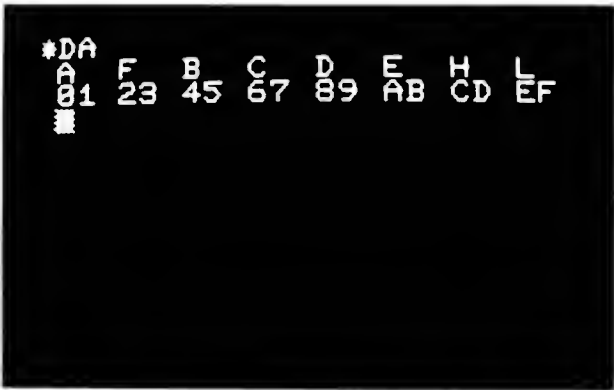
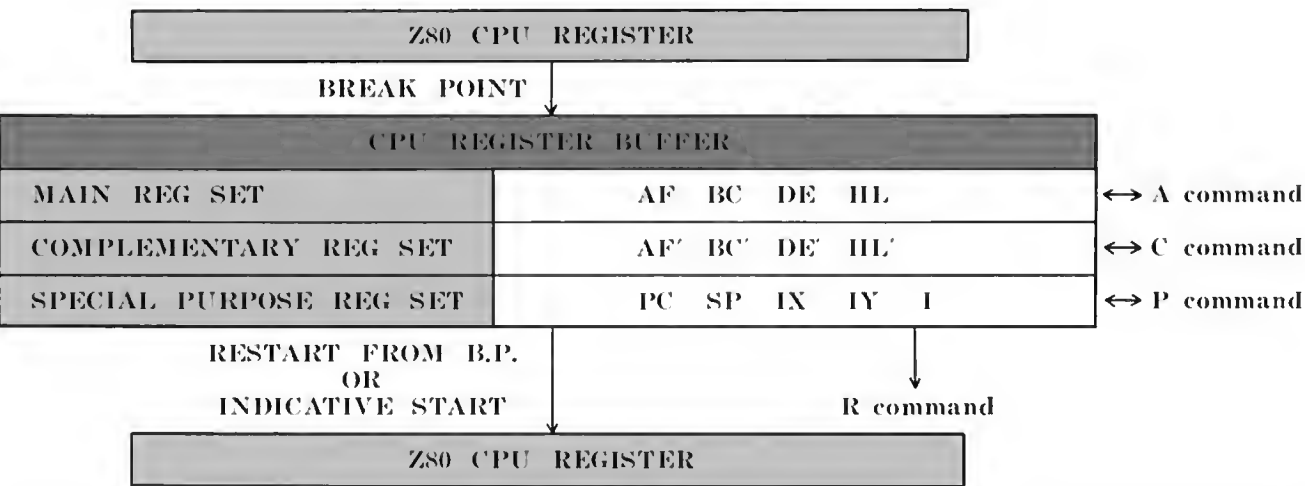
- " *D " のコマンド待ちに、 R (Register) コマンドを与える。
- システムは、CPUレジスタセットの全ての内容を16進 2 桁、または 4 桁で表示する。カーソルは現れず、値を変更することはできない。
ブレイク動作の行われた時点と、 I コマンドによるインディカティブスタートの時点では、この R コマンドと同様のレジスタバッファダンプが自動的に行われる。
- 全レジスタの内容の表示後、次のコマンド待ちに戻る。

レジスタコマンド(A, C, P, R)の利用形態

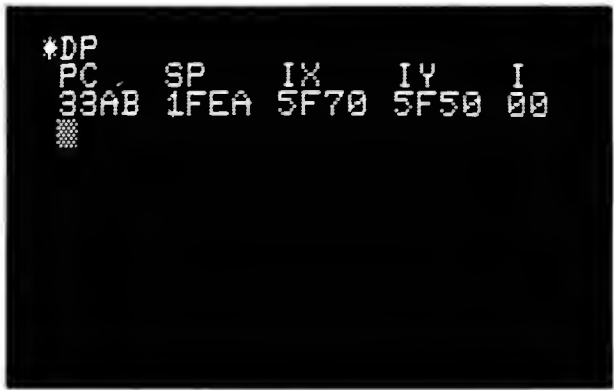
レジスタコマンド (A, C, P, R) によって表示される値は、デバッガ内のレジスタバッファの内容である。デバッガ内のレジスタバッファの内容は、ブレーク動作の行われた時に待避されたCPUレジスタの内容か、または、A、C、Pコマンドのカーソル操作によって変更された値である。

ブレークポイントからのリスタートあるいは、インディカティブスタートは、レジスタバッファの内容をCPUレジスタに復帰することによって行われる。

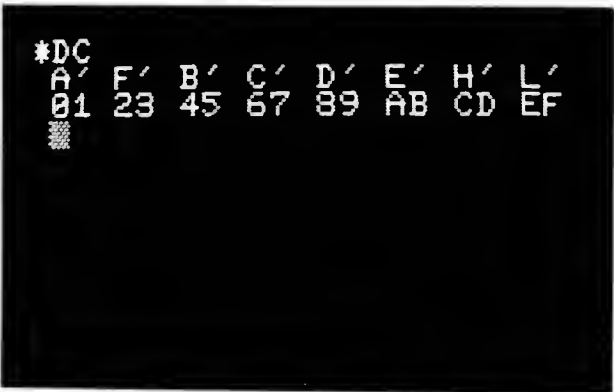
レジスタコマンドの利用形態と、テレビ画面表示例を次に示す。



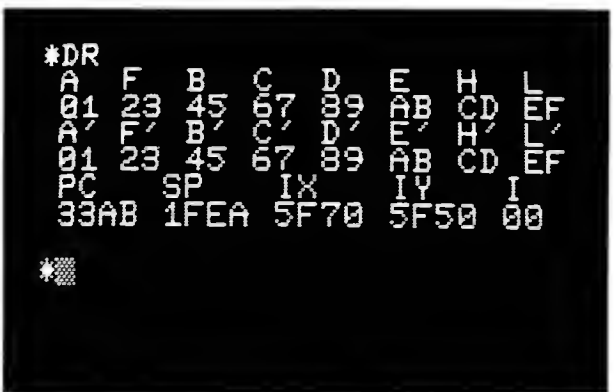
Aコマンド



Pコマンド



Cコマンド



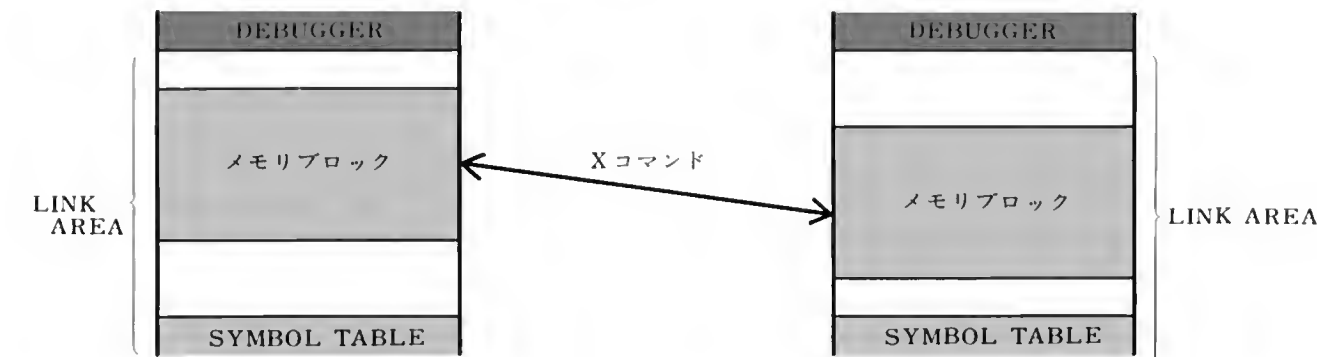
Rコマンド

X(data TRANSfer)コマンド

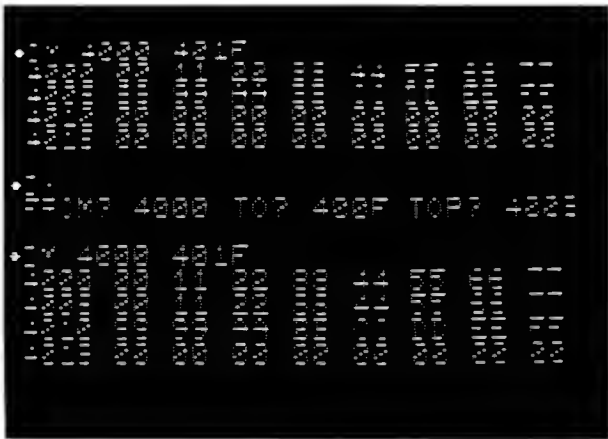
指定されたメモリブロックを、別のメモリエリアへ転送する。

*DX
FROM? 3FF0 TO? 4027 TOP? 5FF0
3FF0から4027までのメモリブロックの内容を5FF0を先頭番地とする
メモリエリアへブロック転送せよ

- " *D " のコマンド待ちに、X (data TRANSfer) コマンドを与える。
- システムは " FROM? " の表示後、転送するメモリブロック先頭アドレスの16進 4 桁入力待ちとなる。
次に " TO? " の表示後、転送するメモリブロックのエンドアドレスの、16進 4 桁入力待ちとなる。さらに、
" TOP? " の表示後、メモリブロックを転送するメモリエリアの先頭アドレスの16進 4 桁入力待ちとなる。
(シンボリックアドレス指定はできない)
- それぞれのアドレスが指定されると、ブロック転送を行い、コマンド待ちに戻る。
- 転送するメモリブロック、転送先のメモリエリアは、LINK AREA内であってはいない。
- 転送するメモリブロックと転送先のメモリエリアが下図に示すように重なる関係になっても、転送されるブ
ロックの内容は壊されずに転送される。(右から左、または左から右の転送いずれの場合も可)



- 右の写真は、4000から400Fまでのメモリブロックを
4008を先頭とするメモリエリアに転送したもようを
示す。
前後のMコマンドによって表示されたメモリ内容を
比較せよ。



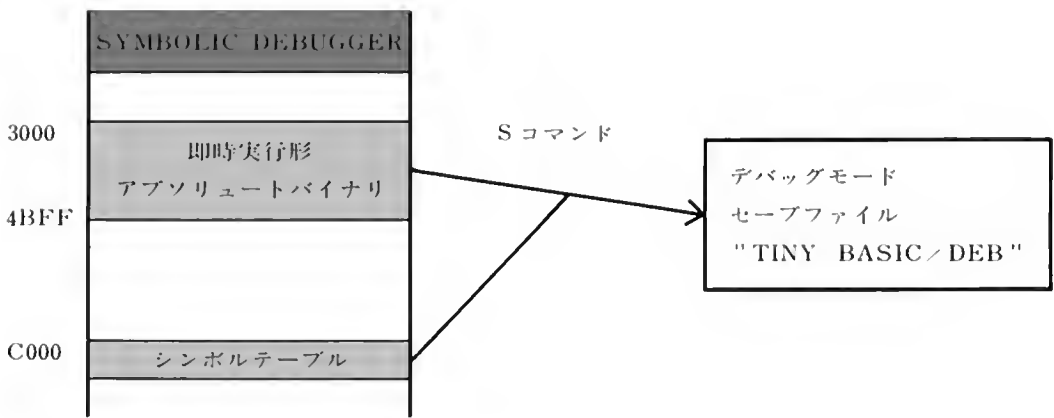
S(Save)コマンド

SYMBOLIC DEBUGGERのリンクエリア内に構成されている、即時実行形のアブソリュートバイナリプログラムに指定されたブロックおよびシンボルテーブルを、ファイルネームを付けた出力ファイルに出力する。従ってこのファイルをYコマンドでロードすると、リンクエリアの内容およびシンボルテーブルが再構成されることになる。

*DS
FILENAME?TINY BASIC/DEB **CR**
FROM? 3000 TO? 4BFF

リンクエリア内の3000から4BFFに構成されている、即時実行形のアブソリュートバイナリブロックとシンボルテーブルとをファイルネーム "TINY BASIC/DEB" を付けてそのまま出力ファイルに出力せよ

- " *D " のコマンド待ちに、S (Save) コマンドを与える。
- システムは改行して " FILENAME? " と表示し、出力ファイル名の指定を待つ。
- ファイルネームを指定したら **CR** を押す。
- 改行して " FROM? " を表示し、出力すべきリンクエリア内の即時実行形のアブソリュートバイナリブロックの先頭アドレスが、16進4桁で指定されるのを待つ。
次に " TO? " を表示し、同じくエンドアドレスの16進4桁指定を待つ。
- 出力するメモリブロックが指定されると、RECORDボタンとPLAYボタンを押すように指示される。
- RECORDおよびPLAYボタンを押すと、指定されたメモリブロックおよび、シンボルテーブルの内容を、出力ファイルに出力する。
- 下図はSコマンドによって、3000から4BFFまでのアブソリュートバイナリブロックを、filename "TINY BASIC /DEB" を付けた出力ファイルに出力するもようを示す。



〔問〕 TEXT EDITORのWコマンド、ASSEMBLERのPASS 4、RELOCATABLE LOADERのSコマンド、そしてSYMBOLIC DEBUGGERのSコマンドによって出力されるそれぞれのファイルの形式、何によってローディングされるか、役割、相違などについて説明せよ。

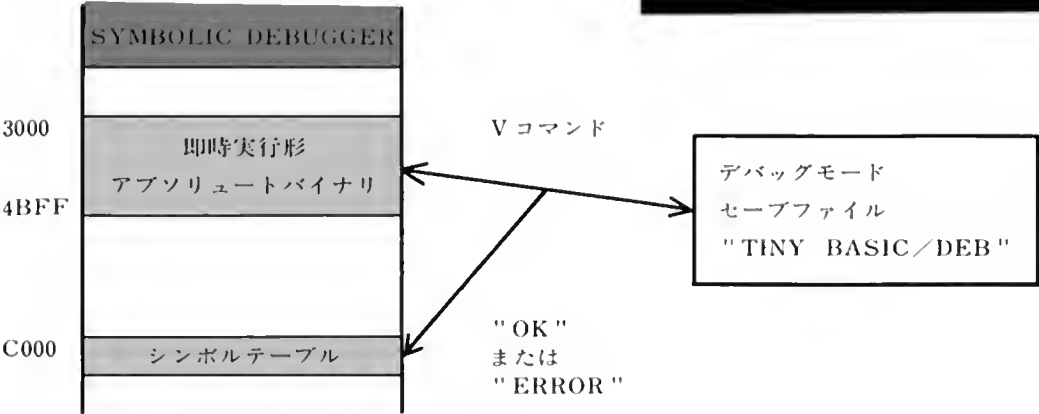
V(Verify)コマンド

ファイルネームで指定されるデバッグモードセーブファイルとリンクエリアの内容とを比較する。

＊DV デバッグモードセーブファイルと、リンクエリアの内容とを比較せよ

- "＊D" のコマンド待ちに、V (Verify) コマンドを与える。
- システムは "FILENAME?" と表示し、比較すべき、デバッグモードセーブファイルの指定を待つ。
- ファイルネームを指定したら **CR** を押す。システムは、PLAY ボタンを押すよう指示する。
- PLAY ボタンを押すと、指定されたファイルを見つけ出し、比較を始める。
ファイルネームを指定していないと、最初に見つけ出したセーブファイルの比較を行う。
- 比較されるファイルと、リンクエリアの内容が同一であれば、"OK" と表示され、相違があれば "ERROR" の表示がなされる。

— 右の写真は、S コマンドによって、デバッグモードセーブファイル "TINY BASIC/DEB" をセーブした後に、V コマンドを実行して、リンクエリアの内容との比較を行ったもようを示している。"OK" 表示によって、リンクエリアの内容が相違なくセーブファイルにコピーされたことを示す。

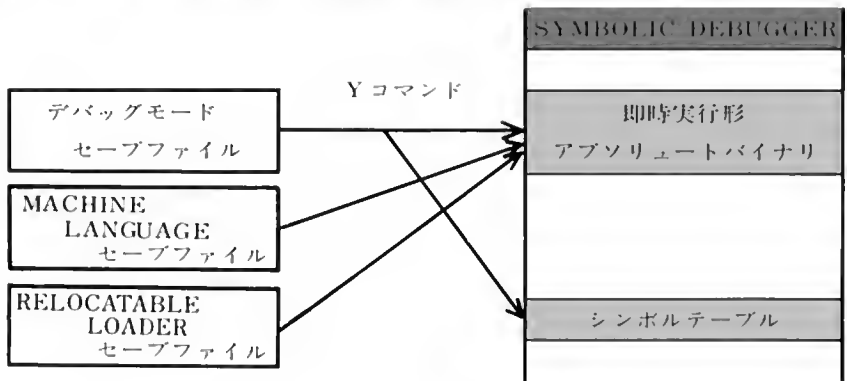


Y(Yank)コマンド

ファイルネームで指定されるデバッグモードセーブファイルを読み込み、リンクエリア内にアブソリュートバイナリプログラムを構成する。

***DY** リンクエリア内をクリアして、デバッグモードセーブファイルを入力せよ
ローディング条件は、ファイルの出力条件と同一とせよ

- " *D " のコマンド待ちに、 Y (Yank) コマンドを与える。
- システムは " FILENAME? " と表示し、読み込むファイルネームの指定を待つ。
- ファイルネームを指定したら **CR** を押す。システムは、PLAYボタンを押すよう指示する。
- PLAYボタンを押すと、指定されたファイルを見つけ出し、読み込みを始める。
ファイルネームを指定していないと、最初に見つけ出したデバッグモードセーブファイルの読み込みを行う。
- ファイル内のアブソリュートバイナリは、S コマンドで出力した時に指定した先頭アドレスからエンドアドレスまでのブロックにそのままストアされる。
- ファイル内のシンボルテーブルの内容は、S コマンドを行なった時の状態がそのまま再現される。但しシンボルテーブルは " *TBL " によって設定されたエリアの先頭からに置きかわる。
- このとき MACHINE LANGUAGE, RELOCATABLE LOADER によって出力されたファイルはシンボルテーブルを持たないので、アブソリュートバイナリファイルだけが読み込まれる。
- 読み込みが終了すると " OK " が表示される。
- 途中でエラーが生じた場合、 " ERROR " を表示する。
- 読み込みを中止する時は、 **SHIFT BREAK** を押す。



- SYMBOLIC DEBUGGERに用意されている、S、V、Y コマンドは、このように、デバッグ操作中のアブソリュートバイナリプログラムを、ファイル化する便宜を図ったものである。デバッグ操作を中断したあと、このファイルを使って、シンボル情報と共に前の状態を再現することができる。
- システムプログラムでは、プログラムの変更、修正は原則として、ソースプログラムの編集に戻ることに
よって行われる。

——特殊コマンド——

コマンド

* D # プリンタへのリスティングのためのリストモードを切り替えよ。

—— " * D " のコマンド待ちに、 # (sharp mark) コマンドを与える。

—— プリンタへのリスティングのため、リストモードが切り替わる。

SYMBOLIC DEBUGGER の起動時は、プリンタリストモードは disable であり、 # コマンドを 1 回与えるごとに、モードは、enable、disable と切り替わる。

プリンタリストモードが enable である時、出力表示は、テレビ画面とプリンタの両方に行われる。

! コマンド

* D ! モニタへコントロールを移せ

—— " * D " のコマンド待ちに、 ! (exclamation mark) を入力する。

—— システムのコントロールは、モニタへ移る。

—— モニタから、SYMBOLIC DEBUGGER へ戻るには、次の 2 通りの方法がある。

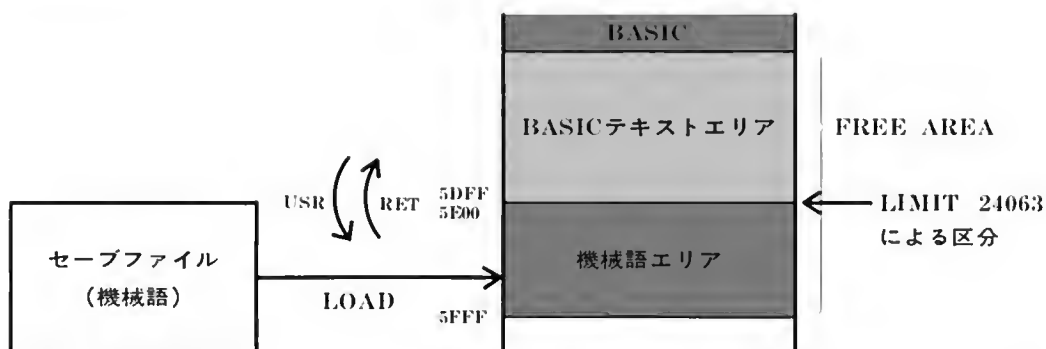
* GOTO \$ 1200 CR リンクエリアの内容をクリアして、スタックを初期状態に戻す。
(COLD START)

* GOTO \$ 1260 CR リンクエリアの内容はクリアせず、コマンド待ちへ移る。
(WARM START)

5 — 4 セーブファイルとBASICテキストとのリンク方法

リロケータブルローダ、またはシンボリックデバッガで出力したセーブファイルをBASICとリンクする方法を述べる。但し、リンクするBASICのバージョンはSP-5010以降のものに限られる。

- リロケータブルローダ、またはシンボリックデバッガで出力するセーブファイルは即時実行可能な機械語プログラムであるが、BASICで、機械語とリンクを行うには、BASICテキスト領域と、機械語をロードするメモリ領域とを区別する必要がある、それはLIMITコマンドによってBASICテキストエリアの下限を指定することによって行う。
- LIMITで区切るアドレス（BASICでは10進表現をとる）は、BASICテキスト、そこで使用する変数のためのバッファエリアを考え、更に、機械語サブルーチンのバイトサイズを考えて指定しなくてはならない。BASICの使用エリアと機械語が重なるようなローディングを行うと"OVERLAY ERROR"となる。
- 機械語セーブファイルのローディング条件は、データアドレスとして指定されたアドレスに従うので、この値が、BASICのLIMITで区切られた機械語エリア内に位置するようにあらかじめアセンブル作業を行っておく必要がある。
- LIMITで区別されたエリアに機械語プログラムがロードされたら、BASICのUSRコマンドを用いて機械語プログラムへコントロールを移すことによってリンクが行える。
USRコマンドは、10進数でアドレスを示すコール命令である。たとえば、"USR (24064)"というのはアセンブリ語で書くとCALL 5E00Hである。
- 下図は、5FFFまでのフリーエリア（RAM20Kバイトの場合に当る）を、"LIMIT 24063"によって5E00番地でBASICエリアと機械語エリアの2つに分けたもようを示す。



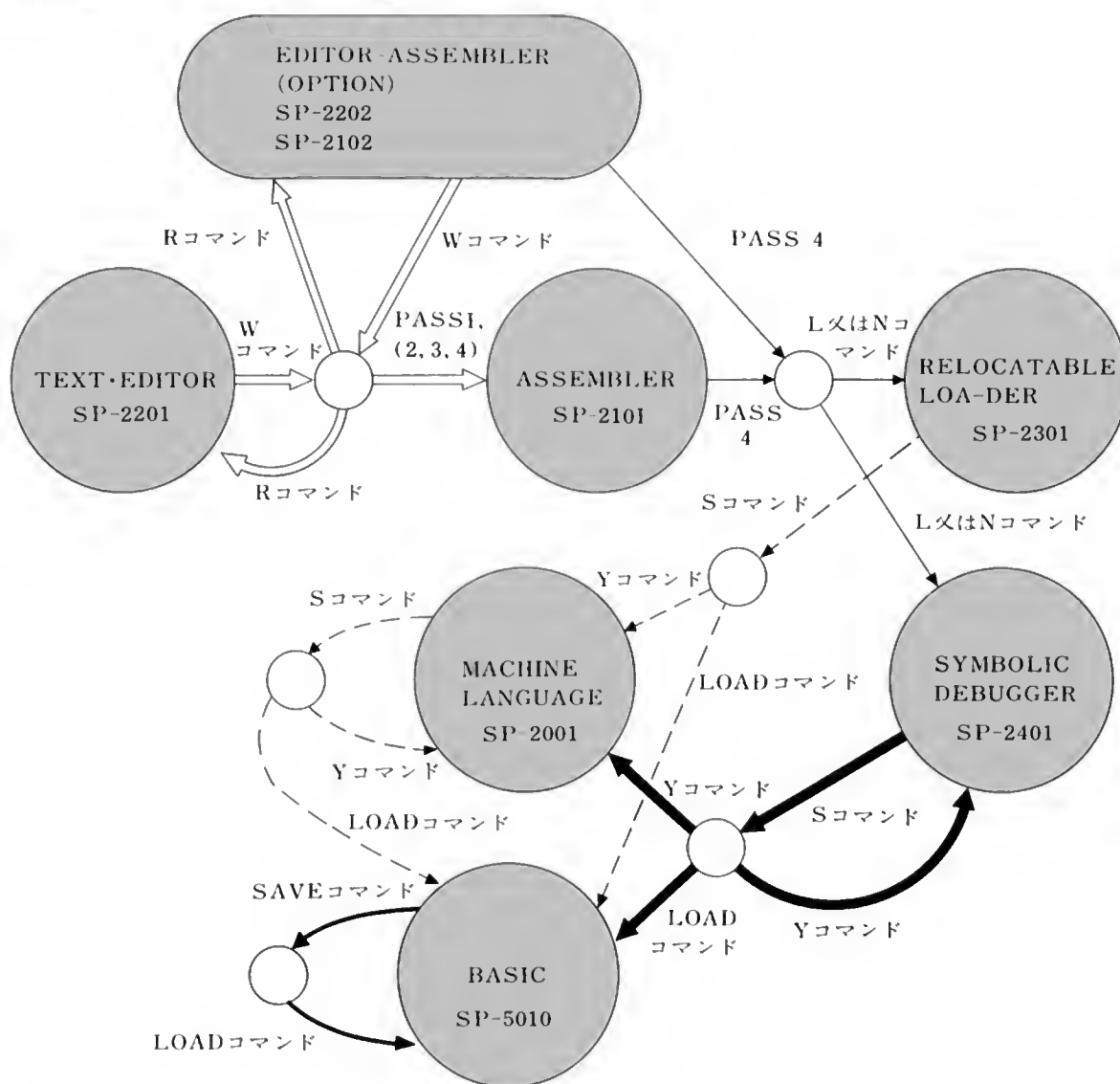
- 右の写真は、BASICプログラムで、セーブファイルをローディングし、USRコマンドでBASICテキストからコントロールを移す例を示している。

```

10 LIMIT 24063
20 LOAD
30 USR(24064)
RUN
↓
PLAY
FOUND GAUSS S-R
LOADING GAUSS S-R
    
```

5 — 5 各システムプログラム間のファイルの考え方

各システムプログラムは必ず1つのファイルを形成して、他のシステムプログラムで利用される。その大きい流れを概念図としてまとめたものが下図である。



——記号の説明——

← ASCII 列によるソースファイル

← リロケートブルバイナリファイル

← セーブファイル（機械語プログラム又は目的ファイル）

← デバッグモードセーブファイル（シンボルテーブル付のセーブファイル）

← BASIC テキストプログラム

○ ファイル媒体（例、外部カセット装置やフロッピーディスクファイルなど）

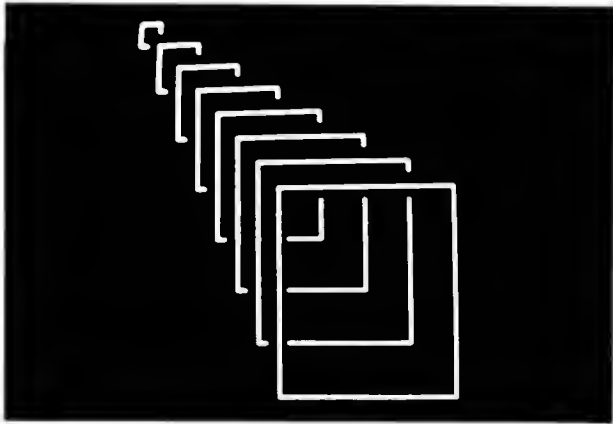
第 6 章

アセンブラプログラミング 例題集

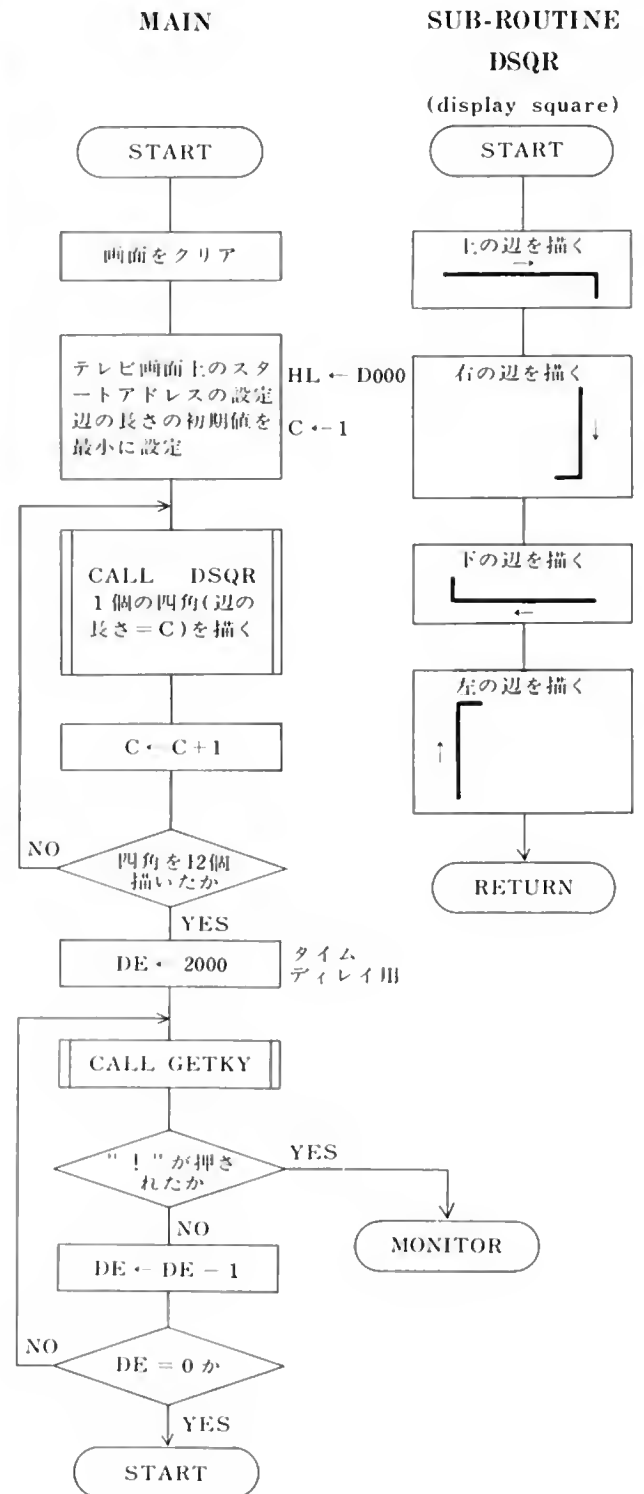
〔注意〕

例題中に示されるプログラム例は、プログラムの構成を示したものである。その中で参照するモニタサブルーチンなどについては、シンボル名を使い、具体的なアドレスを定義していない。従って、プログラムを実際に走らせるためには、リンクする他のプログラムユニット中でその定数を定義するか、あるいは同一のプログラム中に追加して定義する必要がある。未定義シンボル参照の命令には、メッセージ欄に " E " 表示がなされる。

1 接近するに従ってだんだん大きくなる四角形を描いてみる。初めにテレビ画面の左上に小さな四角を描き、それが、近づいて来るようにしたい。それには初めに描いた四角の辺の長さを大きくして行きながら、位置をずらして重ねて描いて行けばよい。



画面の左上からはじめて、四角がだんだん大きくなって近づいて来るように見える。



```

01 0000      ;
02 0000      ;   APPROACHING SQUARE
03 0000      ;
04 0000      REL      2000H
05 2000      ;
06 2000 P     CHR1:    EQU      78H
07 2000 P     CHR2:    EQU      5DH
08 2000 P     CHR3:    EQU      79H
09 2000 P     CHR4:    EQU      1DH
10 2000 P     CHR5:    EQU      1CH
11 2000 P     CHR6:    EQU      5CH
12 2000 P     STADR:   EQU      D000H
13 2000      ;
14 2000 3E16   3DGO:    LD       A,16H
15 2002 CD0000 E   CALL    PRNT
16 2005 112800 LD       DE,0028H
17 2008 2100D0 LD       HL,STADR
18 200B 0E01    LD       C,1
19 200D 41     3DG1:    LD       B,C
20 200E CD3120 CALL    DSQR
21 2011 CD5A20 CALL    TMDLY
22 2014 23     INC      HL
23 2015 19     ADD      HL,DE
24 2016 0C     INC      C
25 2017 79     LD       A,C
26 2018 FE0D   CP       13
27 201A 20F1   JR       NZ,3DG1
28 201C 110020 LD       DE,2000H
29 201F 1B     3DG2:    DEC      DE
30 2020 CD0000 E   CALL    GETKY
31 2023 FE21   CP       21H
32 2025 CA0000 E   JP       Z,MNTR
33 2028 7A     LD       A,D
34 2029 B3     OR       E
35 202A 20F3   JR       NZ,3DG2
36 202C 18D2   JR       3DGO
37 202E      ;
38 202E      ;   SUB ROUTINE
39 202E      ;
40 202E 3E78   DSQR0:   LD       A,CHR1
41 2030 77     LD       (HL),A
42 2031 23     DSQR:    INC      HL
43 2032 10FA   DJNZ     DSQR0
44 2034 3E5D   LD       A,CHR2
45 2036 41     LD       B,C
46 2037 1802   JR       +4
47 2039 3E79   DSQR1:   LD       A,CHR3
48 203B 77     LD       (HL),A
49 203C 19     ADD      HL,DE
50 203D 10FA   DJNZ     DSQR1
51 203F 3E1D   LD       A,CHR4
52 2041 41     LD       B,C
53 2042 1802   JR       +4
54 2044 3E78   DSQR2:   LD       A,CHR1
55 2046 77     LD       (HL),A
56 2047 2B     DEC      HL
57 2048 10FA   DJNZ     DSQR2
58 204A 3E1C   LD       A,CHR5
59 204C 41     LD       B,C
60 204D 1802   JR       +4

```

シンボル	キャラクタ
CHR1	☐
CHR2	▣
CHR3	▢
CHR4	▤
CHR5	▥
CHR6	▦

四角を描くための
ディスプレイコー
ドを定義する。

☐ ビデオ RAM の先頭アドレス。

☐ テレビ画面をクリアする。

☐ 初期値の設定、C←1 は最初の四角の辺の長
さを決める。

☐ C の値に従った四角を 1 個描き、タイムディ
レイをおいて、次の位置を決める (HL)。

☐ 辺の長さを決める C をインクリメントする。
13 になったらこのループを抜け出す。

☐ タイムディレイをとりながら GET KEY を見
る。"! " キーが押されたらモニタへ移る。
その他の場合は、初めに戻る。

☐ 四角の上辺を描く。

☐ 四角の右辺を描く。

☐ 四角の下辺を描く。

*** Z80 ASSEMBLER SP-2101 PAGE 02 ***

01 204F 3E79	DSQR3:	LD	A, CHR3	四角の左辺を描いて RETURN する。
02 2051 77		LD	(HL), A	
03 2052 ED52		SBC	HL, DE	
04 2054 10F9		DJNZ	DSQR3	
05 2056 3E5C		LD	A, CHR6	
06 2058 77		LD	(HL), A	
07 2059 C9		RET		
08 205A	:			DE へレジスタの 2000H 回のタイムディレイ。
09 205A F5	TMDLY:	PUSH	AF	
10 205B D5		PUSH	DE	
11 205C 110020		LD	DE, 2000H	
12 205F 1B		DEC	DE	
13 2060 7A		LD	A, D	
14 2061 B3		OR	E	
15 2062 20FB		JR	NZ, -3	
16 2064 D1		POP	DE	
17 2065 F1		POP	AF	
18 2066 C9		RET		
19 2067		END		

このプログラムでは、モニタサブルーチン "PRNT"、"GETKY" およびモニタ先頭アドレス "MNTR" が参照されているので、他のリンクするプログラムユニット、またはこのプログラムユニットに追加して定義する必要がある。リスト上のメッセージ欄の "E" 表示が外部シンボル参照を意味している。(P.15 参照)

```
PRNT: EQU 0012H
GETKY: EQU 001BH
MNTR: EQU 0000H
```

また四角の辺を描くときに画面がチラつかないようにするために、このリストの第 1 ページの第 41 行、第 48 行、第 55 行、第 2 ページの第 2 行の "LD (HL), A" コマンドの前に、

CALL ? BLNK

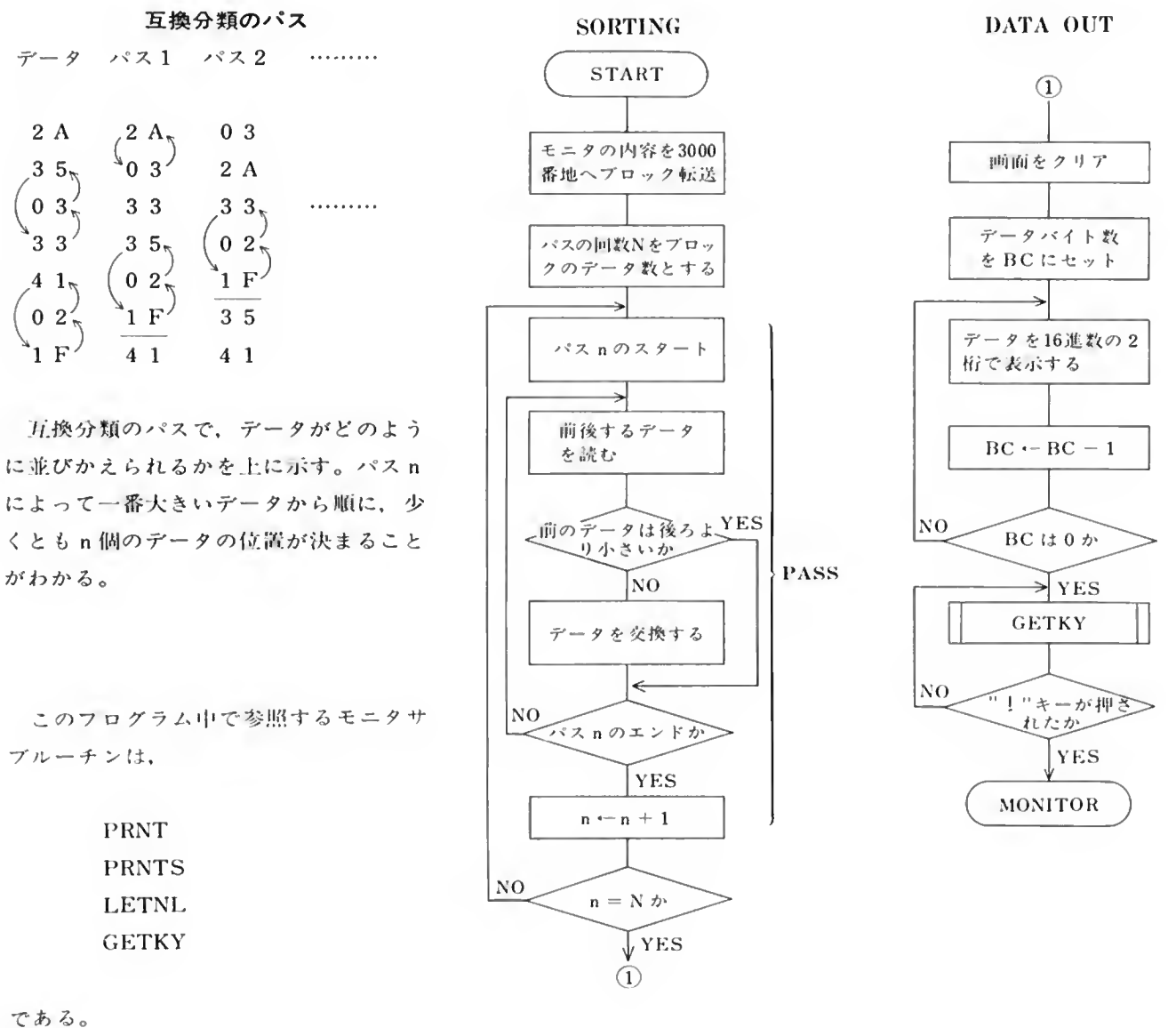
を置いて、垂直ブランキング期間を待つ方法がある。(P.132 参照)

2 メモリのあるブロックの内容をデータと見なして、それを小さい順に分類 (SORTING) する。たとえばモニタの先頭から100 (Hex) バイトの内容をソーティングすることを考える。

モニタの内容を、RAM エリアの3000 番地へブロック転送し、そこで逐次比較による互換分類作業を行う。互換分類法は、隣り合ったデータの大小を比較交換して行く方法であり、ソーティングで最も簡単な方法である。データブロックの先頭から終りまで、順番に互換分類を行って行くと、先ずブロック内の最大のデータが、最後に移ることになる。この1 回の互換分類作業を、パス 1 と呼ぶことにすると、n 個のデータの分類は、最大パス n で完了することになる。今の場合、100 (Hex) 回のパスを行うことになる。

データの分類は、検索 (RETRIEVAL) とともにデータベースの扱いに重要な役割をもつものである。その方法には、幾つかの種類があり、データの形式、大きさ等に従って作業効率が異なる。

モニタ0000 番地から100 (Hex) バイトのデータを RAM エリア (先頭3000 番地) へブロック転送し、分類されたデータをテレビ画面に表示させるプログラムは、次のように構成できる。



** Z80 ASSEMBLER SP-2101 PAGE 01 **

01 0000	:			
02 0000	:	Sorting		
03 0000	:			
04 0000 210000	SORT0:	LD	HL, RDADR	モニタのデータを RAM エリアにブロック転送する。
05 0003 110030		LD	DE, STADR	
06 0006 010F00		LD	BC, BSIZE	
07 0009 EDB0		LDIR		
08 000B 010F00		LD	BC, BSIZE	バスの回数を BC レジスタにセットする。
09 000E 0B		DEC	BC	
10 000F DD210030	SORT1:	LD	IX, STADR	Sorting を行うためにインデックスレジスタ IX を用いる。
11 0013 50		LD	D, B	
12 0014 59		LD	E, C	
13 0015 DD7E00		LD	A, (IX+0)	後ろのデータが前のデータより小さければ交換する。
14 0018 DDBE01		CP	(IX+1)	
15 001B 3809		JR	C, SORT2	
16 001D DD6601		LD	H, (IX+1)	
17 0020 DD7701		LD	(IX+1), A	
18 0023 DD7400		LD	(IX+0), H	
19 0026 DD23	SORT2:	INC	IX	バスを所要回数繰り返すための判別を行う。
20 0028 1B		DEC	DE	
21 0029 7A		LD	A, D	
22 002A B3		OR	E	
23 002B 20E8		JR	NZ, SORT1+6	
24 002D 0B		DEC	BC	
25 002E 78		LD	A, B	
26 002F B1		OR	C	
27 0030 20DD		JR	NZ, SORT1	
28 0032	:			
29 0032	:	HEXA DATA OUT		
30 0032	:			
31 0032 3E16	HOUT0:	LD	A, 16H	画面をクリアする。16H は " " のアスキーコード。
32 0034 CD0000 E		CALL	PRNT	
33 0037 210030		LD	HL, STADR	
34 003A 010F00		LD	BC, BSIZE	
35 003D CD0000 E	HOUT1:	CALL	PRNTS	
36 0040 1602		LD	D, 2	
37 0042 AF	HOUT2:	XOR	A	ロケーション HL の内容の上位 4 ビットを Acc に読み込む。
38 0043 ED6F		RLD		
39 0045 C630		ADD	A, 30H	
40 0047 FE3A		CP	3AH	
41 0049 3802		JR	C, DSPLY	
42 004B C607		ADD	A, 07H	
43 004D CD0000 E	DSPLY:	CALL	PRNT	16進データをアスキーコードに変換して " PR NT " を CALL する。
44 0050 15		DEC	D	
45 0051 20EF		JR	NZ, HOUT2	
46 0053 CD0000 E		CALL	LETNL	Sorting されたデータを全て表示して行く。
47 0056 23		INC	HL	
48 0057 0B		DEC	BC	
49 0058 78		LD	A, B	
50 0059 B1		OR	C	
51 005A 20E1		JR	NZ, HOUT1	
52 005C CD0000 E	KEYIN:	CALL	GETKY	表示を終えたら " ! " のキー入力待ち。(GOTO MONITOR)
53 005F FE21		CP	21H	
54 0061 20F9		JR	NZ, KEYIN	
55 0063 C30000 E		JP	MNTR	
56 0066	:			
57 0066	:	DATA		
58 0066	:			
59 0066 P	RDADR:	EQU	0000H	バイトサイズを変更してみよ。
60 0066 P	STADR:	EQU	3000H	データを大きい順に並べるにはどこを変更すればよいか。
61 0066 P	BSIZE:	EQU	000FH	

RLD Acc

7	4	3	0
---	---	---	---

7	4	3	0
---	---	---	---

 (HL)

3 24時間デジタル時計を作成する。時刻の計数は内蔵時計を読むことによって行う。内蔵時計を読むには、モニタサブルーチン "TIMRD (time read)" を用いる。また時計の起動は "TIMST (time start)" を用いられ

よい。
ここでは内蔵時計は1秒の経過を検出するためだけに使う。時刻の計数は、ビデオRAM中に数値(ディスプレイコード)を置きそれを毎秒カウントアップして行く方法をとってみる。そうすると、テレビ画面への表示を同時に行うことができる。また毎秒"ピッ"と音を鳴らす(あるいは止める)ためのBELLモードも作る。

時刻の設定



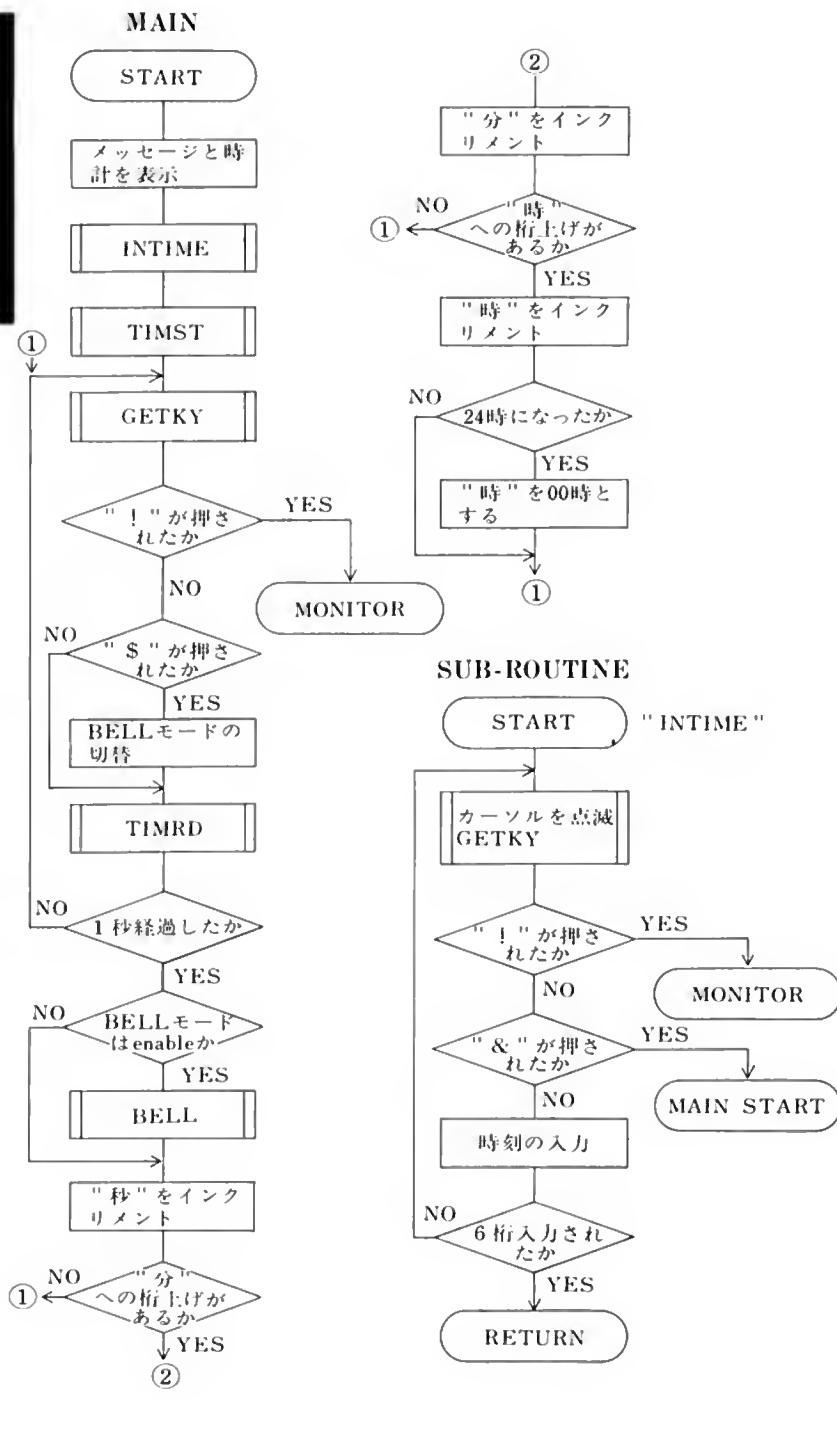
スタートするとカーソルを点滅して現在の時刻を聞いて来る。"時" "分" "秒" それぞれを2桁で入力する。間違った時は"&"キーを押してやり直すことができる。("!"キーを押すとモニタへ戻る)

この時計では、数字以外の文字は入力できないが、時間として不適当な数(25時70分など)のチェックは行っていないので注意。

時計がスタートしたら、1秒ごとに"ピッ"という音がして、時刻が表示される。"\$"キーを押すことによって音を出すモードを切り替えられる。

このプログラム中で参照するモニタサブルーチンは、

TIMST
TIMRD
GETKY
BELL
LETNL
MSG



である。

** Z80 ASSEMBLER SP-2101 PAGE 01 **

01	0000		:				
02	0000		:	DIGITAL CLOCK			
03	0000		:				
04	0000	P	DTH1:	EQU	D1C1H	時、分、秒のそれぞれの桁を表示させる。 V-RAMのアドレスを定義する。	
05	0000	P	DTH0:	EQU	D1C3H		
06	0000	P	DTM1:	EQU	D1C9H		
07	0000	P	DTM0:	EQU	D1CBH		
08	0000	P	DTS1:	EQU	D1D1H		
09	0000	P	DTS0:	EQU	D1D3H		
10	0000		MNTR:	EQU	0000H		
11	0000		REL		2000H		
12	2000		:				
13	2000	310020	START:	LD	SP, START		
14	2003	CDEC20		CALL	MESG0	メッセージと、時計の棒を表示して、時刻の入力を待つ。	
15	2006	CD9220		CALL	INTIME		
16	2009	CD0521		CALL	MESG1		
17	200C	AF		XOR	A	内蔵時計の起動	
18	200D	47		LD	B, A	B<--00としているのはBELLモード切り替えがenableの状態としている。	
19	200E	110000		LD	DE, 0000H		
20	2011	CD0000	E	CALL	TIMST		
21	2014	CD0000	E	CLK0:	CALL	TIMRD	時計の値を読みEの値をCに入れておく。
22	2017	4B		LD	C, E		
23	2018	CD0000	E	CALL	GETKY	キーを取り込む。	
24	201B	FE21		CP	21H	" ! " (21H) キーが押されているとモニタへジャンプする。	
25	201D	CA0000		JP	Z, MNTR	" \$ " (24H) キーが押されているとBELLモード切り替えへ移る。	
26	2020	FE24		CP	24H		
27	2022	2804		JR	Z, +6		
28	2024	0600		LD	B, 0H	BELLモード切り替えをenableとする。	
29	2026	180C		JR	CLK1		
30	2028	A0		AND	B	BELLモードが切り替った状態(B<--FF)では続けて、モード切り替えを行わない(disable)。	
31	2029	2009		JR	NZ, +11	BELLのモードを切り替える。	
32	202B	3AC121		LD	A, (PIPP1)	(PIPP1) を00→FF, またはFF→00とする。	
33	202E	2F		CPL			
34	202F	32C121		LD	(PIPP1), A		
35	2032	06FF		LD	B, FFH	BELLモードが切り替った状態をBレジスタに設定。	
36	2034	CD0000	E	CLK1:	CALL	TIMRD	時計の値が変化したかどうか(1秒たったかどうか)調べる。
37	2037	7B		LD	A, E		
38	2038	B9		CP	C		
39	2039	28DD		JR	Z, CLK0+4		
40	203B	3AC121		LD	A, (PIPP1)	(PIPP1) が00ならば BELL を鳴らす。	
41	203E	B7		OR	A		
42	203F	2003		JR	NZ, +5		
43	2041	CD0000	E	CALL	BELL		
44	2044	21D3D1		LD	HL, DTS0	" 秒 " の 1 の位の値を読み込み(ディスプレイコード)インクリメントする。	
45	2047	7E		LD	A, (HL)	桁上げがあれば値を0にして、10の位をインクリメントする。	
46	2048	3C		INC	A	桁上げがなければ時計の読み込み(CLR0)へ戻る。	
47	2049	FE2A		CP	2AH		
48	204B	2803		JR	Z, CLK2		
49	204D	77	JRCLK0:	LD	(HL), A		
50	204E	18C4		JR	CLK0		
51	2050	CD1321	CLK2:	CALL	SETZR		
52	2053	7E		LD	A, (HL)		
53	2054	3C		INC	A		
54	2055	FE26		CP	26H	" 秒 " がインクリメントされて60秒になったら, " 分 " に桁上げする。	
55	2057	20F4		JR	NZ, JRCLK0		
56	2059	3E20		LD	A, 20H		
57	205B	77		LD	(HL), A		
58	205C	21CBD1		LD	HL, DTM0		
59	205F	7E		LD	A, (HL)		
60	2060	3C		INC	A		
61	2061	FE2A		CP	2AH	" 分 " の 1 の位で桁上げがあるか調べる。	
62	2063	20E8		JR	NZ, JRCLK0		

** Z80 ASSEMBLER SP-2101 PAGE 02 **

01 2065	CD1321	CALL	SETZR	桁上げがあれば10の位をインクリメントする。
02 2068	7E	LD	A, (HL)	
03 2069	3C	INC	A	"分"がインクリメントされて60分になったら"時"に桁上げする。
04 206A	FE26	CP	26H	
05 206C	20DF	JR	NZ, JRCLKO	"時"がインクリメントされて24時になったら00時に戻す。
06 206E	3E20	LD	A, 20H	
07 2070	77	LD	(HL), A	
08 2071	21C3D1	LD	HL, DTHO	
09 2074	7E	LD	A, (HL)	
10 2075	3C	INC	A	
11 2076	FE24	CP	24H	
12 2078	200E	JR	NZ, CLK3	
13 207A	2B	DEC	HL	
14 207B	2B	DEC	HL	
15 207C	7E	LD	A, (HL)	
16 207D	23	INC	HL	
17 207E	23	INC	HL	
18 207F	FE22	CP	22H	
19 2081	2005	JR	NZ, CLK3	
20 2083	CD1321	CALL	SETZR	
21 2086	18C5	JR	JRCLKO	
22 2088	FE2A	CP	2AH	
23 208A	20C1	JR	NZ, JRCLKO	
24 208C	CD1321	CALL	SETZR	
25 208F	34	INC	(HL)	
26 2090	18BC	JR	JRCLKO+1	
27 2092				
28 2092				
29 2092				
30 2092	21C1D1	INTIME: LD	HL, DTH1	
31 2095	CDA120	CALL	INPUT	"時" "分" "秒"の順に現在時刻の入力を待つ。
32 2098	21C9D1	LD	HL, DTM1	
33 209B	CDA120	CALL	INPUT	
34 209E	21D1D1	LD	HL, DTS1	
35 20A1				
36 20A1	1E02	INPUT: LD	E, 02H	
37 20A3	CDC120	CALL	KEYIN	それぞれ10の位、1の位の2桁ずつ入力するためにEに2を入れる。
38 20A6	FE21	CP	21H	
39 20A8	CA0000	JP	Z, MNTR	"! " (21H) キーが入力されるとモニタへ、"& " (26H) キーが入力されるとスタートへ戻る。
40 20AB	FE26	CP	26H	
41 20AD	CA0020	JP	Z, START	0から9以外の文字は入力されない。(キー入力はアスキーコードによる)
42 20B0	FE30	CP	30H	
43 20B2	38EF	JR	C, INPUT+2	アスキーコードをディスプレイコードに変換してV-RAMの、時計表示の桁へセットする。
44 20B4	FE3A	CP	3AH	
45 20B6	30EB	JR	NC, INPUT+2	
46 20B8	E62F	AND	2FH	
47 20BA	77	LD	(HL), A	
48 20BB	23	INC	HL	
49 20BC	23	INC	HL	
50 20BD	1D	DEC	E	
51 20BE	20E3	JR	NZ, INPUT+2	
52 20C0	C9	RET		
53 20C1				
54 20C1	36EF	KEYIN: LD	(HL), EFH	
55 20C3	CDCE20	CALL	1KEY	カーソル(EFコード)を点滅しながら1文字のキー入力を待つ。
56 20C6	C0	RET	NZ	
57 20C7	77	LD	(HL), A	
58 20C8	CDCE20	CALL	1KEY	
59 20CB	C0	RET	NZ	
60 20CC	18F3	JR	KEYIN	

01 20CE	;			キー入力がない時80回ループして戻るため
02 20CE 0680	1KEY:	LD	B, 80H	めの値。
03 20D0 CD0000 E		CALL	GETKY	Accに取り込まれた値をCにセット。
04 20D3 4F		LD	C, A	
05 20D4 AF		XOR	A	チャタリング防止のためのタイムディレイ。
06 20D5 3D		DEC	A	
07 20D6 20FD		JR	NZ, -1	
08 20D8 CD0000 E		CALL	GETKY	
09 20DB B9		CP	C	もう一度 GETKY を行い、チャタリングを
10 20DC 20F2		JR	NZ, 1KEY+2	防止する。
11 20DE B7		OR	A	
12 20DF 2808		JR	Z, +10	Accが00かどうか調べる。
13 20E1 CD0000 E		CALL	GETKY	
14 20E4 B7		OR	A	Accが00でないなら、キーが離されるのを待
15 20E5 20FA		JR	NZ, -4	って、OR Cを行いキー入力された値をAccに
16 20E7 B1		OR	C	戻し、ZフラグをリセットしてRETURNする。
17 20E8 C9		RET		
18 20E9 10E5		DJNZ	1KEY+2	
19 20EB C9		RET		
20 20EC	;			
21 20EC 111921	MESGO:	LD	DE, DATA1	メッセージおよび、時計の枠のデータを表示
22 20EF CD0C21		CALL	NLMSG	するサブルーチン。
23 20F2 113F21		LD	DE, DATA2	
24 20F5 CD0C21		CALL	NLMSG	
25 20F8 115D21		LD	DE, DATA3	
26 20FB CD0C21		CALL	NLMSG	
27 20FE 117D21		LD	DE, DATA4	
28 2101 CD0C21		CALL	NLMSG	
29 2104 C9		RET		
30 2105	;			
31 2105 119B21	MESG1:	LD	DE, DATA5	DATA5のメッセージ「タダイマノジコクハ」
32 2108 CD0C21		CALL	NLMSG	を表示するサブルーチン。
33 210B C9		RET		
34 210C	;			
35 210C CD0000 E	NLMSG:	CALL	LETNL	改行し、メッセージアウトするためのサブル
36 210F CD0000 E		CALL	MSG	ーチン。
37 2112 C9		RET		
38 2113	;			
39 2113 3E20	SETZR:	LD	A, 20H	ディスプレイコード29がインクリメントされ
40 2115 77		LD	(HL), A	て2Aになった時、桁上げをするためにコード
41 2116 2B		DEC	HL	を20に戻して、10の桁へポインタを移すた
42 2117 2B		DEC	HL	めのサブルーチン。
43 2118 C9		RET		
44 2119	;			
45 2119	; DATA			
46 2119	;			
47 2119 1611	DATA1:	DEFW	1116H	クリアホームしてカーソルを9回下げるため
48 211B 1111		DEFW	1111H	のカーソルコントロール用アスキーコード。
49 211D 1111		DEFW	1111H	
50 211F 1111		DEFW	1111H	
51 2121 1111		DEFW	1111H	
52 2123 99BE9B		DEFM	'ゲンザイ ノ ジコク ヲ セット シテクダサイ'	
53 2127 BE9220A9				
54 212B 209CBE9A				
55 212F 98208620				
56 2133 9E8FA420				
57 2137 9CA398A0				
58 213B BE9B92				
59 213E 0D		DEFB	0DH	

** Z80 ASSEMBLER SP-2101 PAGE 04 **

```

01 213F      ;
02 213F 20202020 DATA2:  DEFM  '      _ _ _ _ _ '
03 2143 20202020
04 2147 DOEOD2EO
05 214B CE202020
06 214F DOEOD2EO
07 2153 CE202020
08 2157 DOEOD2EO
09 215B CE
10 215C OD      DEFB  ODH
11 215D      ;
12 215D 20202020 DATA3:  DEFM  '      | 時  |  | 分  |  | 秒  '
13 2161 20202020
14 2165 FD202020
15 2169 FD207920
16 216D FD202020
17 2171 FD207A20
18 2175 FD202020
19 2179 FD207B
20 217C OD      DEFB  ODH
21 217D      ;
22 217D 20202020 DATA4:  DEFM  '      _ _ _ _ _ '
23 2181 20202020
24 2185 CDEOD1EO
25 2189 DD202020
26 218D CDEOD1EO
27 2191 DD202020
28 2195 CDEOD1EO
29 2199 DD
30 219A OD      DEFB  ODH
31 219B      ;
32 219B 1511     DATA5:  DEFW  1115H
33 219D 1111     DEFW  1111H
34 219F 1111     DEFW  1111H
35 21A1 1111     DEFW  1111H
36 21A3 1111     DEFW  1111H
37 21A5 AOA0BE92 DEFM  'タダイマ ノ ジコク ハ?'
38 21A9 AF20A920
39 21AD 9CBE9A98
40 21B1 20AA3F20
41 21B5 20202020
42 21B9 20202020
43 21BD 202020
44 21C0 OD      DEFB  ODH
45 21C1      ;
46 21C1 00     PIPPI:  DEFB  00H
47 21C2      END

```

ホームをして、カーソルを9回下げるための
カーソルコントロール用アスキーコード。

☐ 音を毎秒ビッピッと鳴らすかどうかのモード
を入れておくためのバッファ。

〔応用〕 この時計に、タイマー機能を付け加えよ。また " 25時63分80秒 " などと入力されたら " 02時04分20秒 " と変換するようにしてみよ。

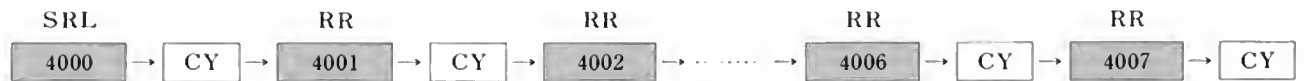
〔応用〕 BASIC の TIS は、モニタ サブルーチン " TIMRD " によって D E レジスタに読み出された秒単位の時間を、時、分、移の6桁に変換したストリングである。

" PRINT TIS " とキー入力すると、BASIC と同じ6桁の時刻表示が得られるプログラムを考えよ。

4

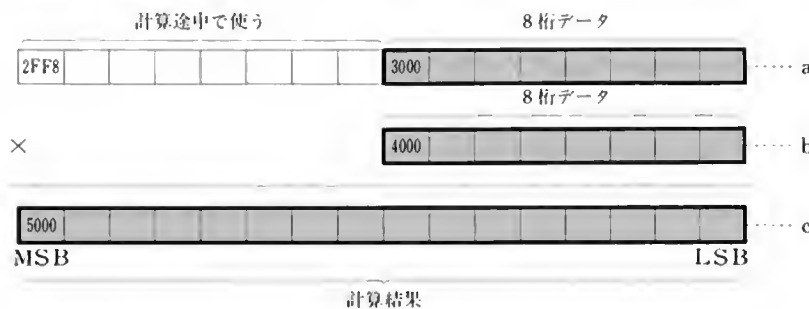
8桁の16進数の乗算を行うプログラムを作成する。乗算を実行するには、データのシフトコマンドが活躍する。たとえば、2進数、00010100 (14H) の00000100 (04H) 倍とは、00010100を左へ2回シフトしたデータ01010000 (50H) が答である。これは、1バイトの(1桁の)16進数の乗算の場合である。

8桁の16進数については、8バイトのメモリエリアを用いて、そこでシフト、ローテート命令を用いて一連の8桁のデータのシフトを実行させる。たとえば、4000から4007番地に8桁のデータがある場合、全体を1ビット右へシフトするには、SRLコマンドとRRコマンドを用いて



で実行される。この作業は、 $a \times b$ の演算で、 b の値(バイナリ)をビット単位で読む時に使う。即ち上で、最後のRRコマンドを実行したときのCY(つまり一番右のCY)が1であれば、 a の値を必要回数左へシフトして行き $a \times b$ の答は、そうしてシフトされた a の値の総和となる。

演算に使用するメモリエリアを次のようにとる。



8桁の16進数 a 、 b を、それぞれ3000、4000番地からの8バイトにセットしておき、演算結果 c ($c = a \times b$)を5000番地からの16バイトにセットするようにする。

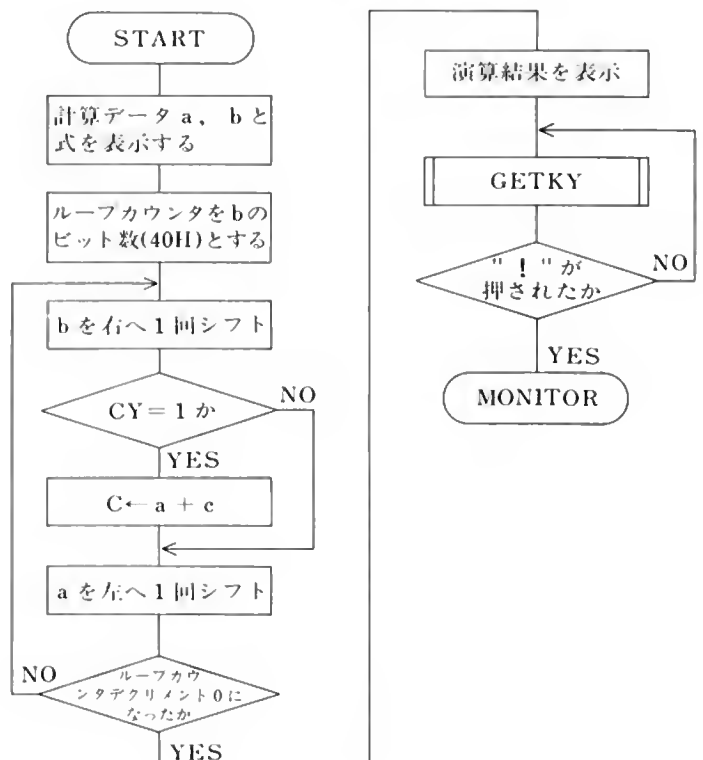
それぞれのデータ番地を次のシンボルに定義する。

- a の先頭番地→MLTPR (3000H)
- b の先頭番地→MLTCD (4000H)
- c の最終番地→RSLT (500FH)

テレビ画面への表示は、8桁の a 、 b 、乗算記号、横線、それに答の c が16進表示される。

$$\begin{array}{r} \text{a a a a a a a a} \\ \times \quad \text{b b b b b b b b} \\ \hline \text{c c c c c c c c c c c c c c} \end{array}$$

プログラムのフローチャートを右図に示す。



```

01 0000      ;
02 0000      ;   UNSIGNED 8 BYTES BINARY MULTIPLY
03 0000      ;
04 0000 P    LOOP:   EQU    40H
05 0000 P    MLTPR:  EQU    3000H
06 0000 P    MLTCD:  EQU    4000H
07 0000 P    RSLT:   EQU    500FH
08 0000 P    DADR1:  EQU    D17BH
09 0000 P    DADR2:  EQU    D1C6H
10 0000 P    DADR3:  EQU    D20AH
11 0000 P    DADR4:  EQU    D25BH
12 0000 P    MNTR:   EQU    0000H
13 0000      REL    2000H
14 2000      ;
15 2000 11C020   START: LD    DE,MSSG
16 2003 CD0000   E    CALL  MSG
17 2006 CD6020   CALL  DISPO
18 2009 CDA820   CALL  CLR
19 200C 0E40     LD    C,LOOP
20 200E FD210040 MULTO: LD    IY,MLTCD
21 2012 0607     LD    B,07H
22 2014 FDCB003E SRL    (IY+0)
23 2018 FD23     INC    IY
24 201A FDCB001E RR      (IY+0)
25 201E 10F8     DJNZ  -6
26 2020 3014     JR     NC,MULT2
27 2022 AF       XOR    A
28 2023 DD210030 LD    IX,MLTPR
29 2027 210F50   LD    HL,RSLT
30 202A 0610     LD    B,10H
31 202C DD7E07   MULT1: LD    A,(IX+7)
32 202F 8E       ADC    A,(HL)
33 2030 77       LD    (HL),A
34 2031 DD2B     DEC    IX
35 2033 2B       DEC    HL
36 2034 10F6     DJNZ  MULT1
37 2036 DD210030 MULT2: LD    IX,MLTPR
38 203A 060F     LD    B,0FH
39 203C DDCB0726 SLA    (IX+7)
40 2040 DD2B     DEC    IX
41 2042 DDCB0716 RL      (IX+7)
42 2046 10FB     DJNZ  -6
43 2048 0D       DEC    C
44 2049 20C3     JR     NZ,MULTO
45 204B 210050   LD    HL,5000H
46 204E 115BD2   LD    DE,DADR4
47 2051 0610     LD    B,10H
48 2053 CD8820   CALL  DISPl+2
49 2056 CD0000   E 1KEYG: CALL  GETKY
50 2059 FE21     CP     21H
51 205B CA0000   JP     Z,MNTR
52 205E 18F6     JR     1KEYG
53 2060      ;
54 2060      ;   SUB-ROUTINE
55 2060      ;
56 2060 210030   DISPO: LD    HL,MLTPR
57 2063 117BD1   LD    DE,DADR1
58 2066 CD8620   CALL  DISPl
59 2069 210040   LD    HL,MLTCD
60 206C 11C6D1   LD    DE,DADR2

```

ループカウンタ
計算データおよび結果の入るメモリの先頭番地。
ディスプレイを行う画面位置の設定。

タイトルを表示する。
計算式を表示する。
メモリのクリアとループカウンタの設定→Cレジ
乗算式 $a \times b = c$ の b に相当するデータ (4000番地からの8バイト) を右へシフトする。
キャリフラグ CFをチェックして、CF=1であればaとcとを加算する。
a, b, cはそれぞれIX, IY, HLの各ペアレジスタでデータのアドレスを指定している。

aの値を左へシフトする。
ループカウンタに指定した回数繰り返す。
計算結果を表示する。
GET KEYを行い"!"が入力されたらモニタへ戻る。

計算式を表示するサブルーチン。


```

01 206F 3E6D          LD      A,6DH
02 2071 12            LD      (DE),A
03 2072 13            INC     DE
04 2073 13            INC     DE
05 2074 13            INC     DE
06 2075 13            INC     DE
07 2076 13            INC     DE
08 2077 CD8620        CALL    DISP1
09 207A 110AD2        LD      DE,DADR3
10 207D 3E78          LD      A,78H
11 207F 0622          LD      B,22H
12 2081 12            LD      (DE),A
13 2082 13            INC     DE
14 2083 10FC          DJNZ    -2
15 2085 C9            RET
16 2086                ;
17 2086                ;
18 2086 0608          DISP1: LD      B,08H
19 2088 7E            LD      A,(HL)
20 2089 23            INC     HL
21 208A 4F            LD      C,A
22 208B CD9520        CALL    DISP2
23 208E 79            LD      A,C
24 208F CD9920        CALL    DISP3
25 2092 10F4          DJNZ    DISP1+2
26 2094 C9            RET
27 2095                ;
28 2095                ;
29 2095 1F            DISP2: RRA
30 2096 1F            RRA
31 2097 1F            RRA
32 2098 1F            RRA
33 2099 E60F          DISP3: AND     0FH
34 209B FE0A          CP      0AH
35 209D 3004          JR      NC,+6
36 209F C620          ADD     A,20H
37 20A1 1802          JR      +4
38 20A3 D609          SUB     09H
39 20A5 12            LD      (DE),A
40 20A6 13            INC     DE
41 20A7 C9            RET
42 20A8                ;
43 20A8                ;
44 20A8 DD210030      CLR:   LD      IX,MLTPR
45 20AC 0608          LD      B,08H
46 20AE AF            XOR     A
47 20AF DD77FF        LD      (IX-1),A
48 20B2 DD2B          DEC     IX
49 20B4 10F9          DJNZ    -5
50 20B6 210F50        LD      HL,RSLT
51 20B9 0610          LD      B,10H
52 20BB 77            LD      (HL),A
53 20BC 2B            DEC     HL
54 20BD 10FC          DJNZ    -2
55 20BF C9            RET
56 20C0                ;
57 20C0                ; MESSAGE DATA
58 20C0                ;
59 20C0 16            MSSG:  DEFB    16H
60 20C1 1111          DEFW    1111H

```

ディスプレイコード4DHのキャラクタは✕

ディスプレイコード78Hのキャラクタは≡

HLレジで指定される1バイトデータをディスプレイコードに変換して、DEレジで示すV-RAMに書き込んでいる。

この2行の処理を変えればアスキーコードにも変換できる。

2FF8～2FFF番地
5000～500F番地
をクリアする。

タイトル表示位置までのカーソル移動コード。

```

01 20C3 1313      DEFW 1313H
02 20C5 554E      DEFW 4E55H
03 20C7 5349      DEFW 4953H
04 20C9 474E      DEFW 4E47H
05 20CB 4544      DEFW 4445H
06 20CD 2038      DEFW 3820H
07 20CF 2042      DEFW 4220H
08 20D1 5954      DEFW 5459H
09 20D3 4553      DEFW 5345H
10 20D5 2042      DEFW 4220H
11 20D7 494E      DEFW 4E49H
12 20D9 4152      DEFW 5241H
13 20DB 5920      DEFW 2059H
14 20DD 4D55      DEFW 554DH
15 20DF 4C54      DEFW 544CH
16 20E1 4950      DEFW 5049H
17 20E3 4C59      DEFW 594CH
18 20E5 0D        DEFB 0D
19 20E6           END
    
```

メッセージデータ。
メッセージは、

'UNSIGNED 8 BYTES BINARY
MULTIPLY'

である。
DEFMを用いて一挙に書くことも可能。

メッセージのエンドマーク。

このプログラムではモニタサブルーチン "MSG", "GETKY" がコールされている。

〔応用〕 シンボリックデバッガの、W (memory Write) コマンドを用いて、乗算データ (3000からの8バイトと4000からの8バイト) をいろいろ変更して、乗算を行ってみよ。

〔応用〕 モニタサブルーチン "GETL" を使って、カーソル操作で乗算式を書くようにせよ。もちろん、乗算は式に与えられた数値について行う。

〔応用〕 10進数の乗算を行うようにするには、どこを変えればよいか考えよ。また10進4則演算がすべて行えるような拡張を考えよ。

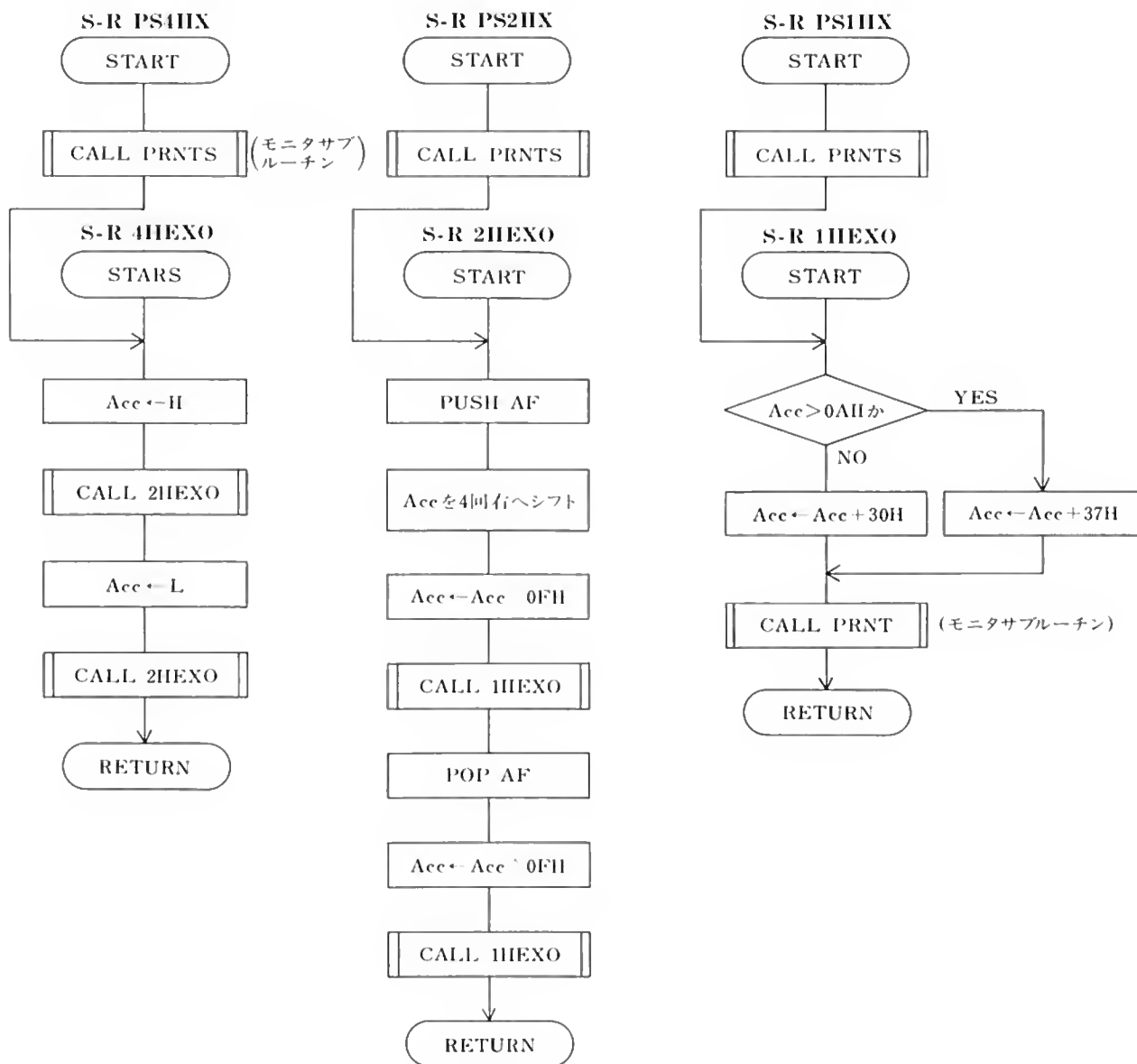
5

2進データを16進表示するサブルーチンを作成する。データは、HLレジスタの内容の16進4桁表示、アキュムレータの内容の16進2桁表示、アキュムレータの下位4ビットの16進1桁表示の3通できるようにする。また表示の前に、スペースを1個表示することもできるようにする。

サブルーチンコールを次の6通りとする。

CALL 4HEX0	(4hexa data out)	HLの内容を表示
CALL PS4HX	(print space, 4hexa data out)	スペースおよびHLの内容を表示
CALL 2HEX0	(2hexa data out)	Accの内容を表示
CALL PS2HX	(print space, 2hexa data out)	スペースおよびAccの内容を表示
CALL 1HEX0	(1hexa data out)	Acc下位4ビットの内容を表示
CALL PS1HX	(print space, 1hexa data out)	スペースおよびAcc下位4ビットの内容を表示

これらのサブルーチンは互いに関連があり、"4HEX0"は"2HEX0"を2回CALLし、"2HEX0"は、"1HEX0"を2回CALLして次のように構成することができる。



HEXA DATA OUT SUB-ROUTINE

** Z80 ASSEMBLER SP-2101 PAGE 01 **

```

01 0000      ;
02 0000      ; 4 HEXA DATA OUT (DESTROYED:A)
03 0000      ; CALL 4HEXO
04 0000      ; CALL PS4HX (PRINT SPACE)
05 0000      ;
06 0000      PS4HX: ENT
07 0000 F5      PUSH AF
08 0001 CD0000 E CALL PRNTS
09 0004 F1      POP AF
10 0005      4HEXO: ENT
11 0005 7C      LD A, H
12 0006 CD1300  CALL 2HEXO
13 0009 7D      LD A, L
14 000A CD1300  CALL 2HEXO
15 000D C9      RET
16 000E      SKP 2

```

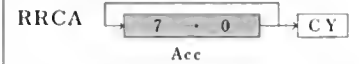
H, LをそれぞれAccに入れて2HEXOをCALLしている。

```

19 000E      ;
20 000E      ; 2 HEXA DATA OUT (DESTROYED:A)
21 000E      ; CALL 2HEXO
22 000E      ; CALL PS2HX
23 000E      ;
24 000E      PS2HX: ENT
25 000E F5      PUSH AF
26 000F CD0000 E CALL PRNTS
27 0012 F1      POP AF
28 0013      2HEXO: ENT
29 0013 F5      PUSH AF
30 0014 0F      RRCA
31 0015 0F      RRCA
32 0016 0F      RRCA
33 0017 0F      RRCA
34 0018 E60F    AND OFH
35 001A CD2900  CALL 1HEXO
36 001D F1      POP AF
37 001E E60F    AND OFH
38 0020 CD2900  CALL 1HEXO
39 0023 C9      RET
40 0024      SKP 2

```

Accの内容を4回右へローテート



Accの下位の4ビットについて2回CALL 1HEXOをしている。

```

43 0024      ;
44 0024      ; 1 HEXA DATA OUT (DESTROYED:A)
45 0024      ; CALL 1HEXO
46 0024      ; CALL PS1HX
47 0024      ;
48 0024      PS1HX: ENT
49 0024 F5      PUSH AF
50 0025 CD0000 E CALL PRNTS
51 0028 F1      POP AF
52 0029      1HEXO: ENT
53 0029 FE0A    CP OAH
54 002B 3006    JR NC, HXT1
55 002D C630    ADD A, 30H
56 002F CD0000 E HXT0: CALL PRNT
57 0032 C9      RET
58 0033 C637    HXT1: ADD A, 37H
59 0035 18F8    JR HXT0
60 0037      END

```

0から9までは30を加えて数字のASCIIコードを作る

AからFまでは、37を加えてアルファベットのASCIIコードを作る。

このサブルーチンを動かすのには、他のプログラムユニット中で、モニタサブルーチン " PRNTS " " PRNT " が定義されている必要がある。

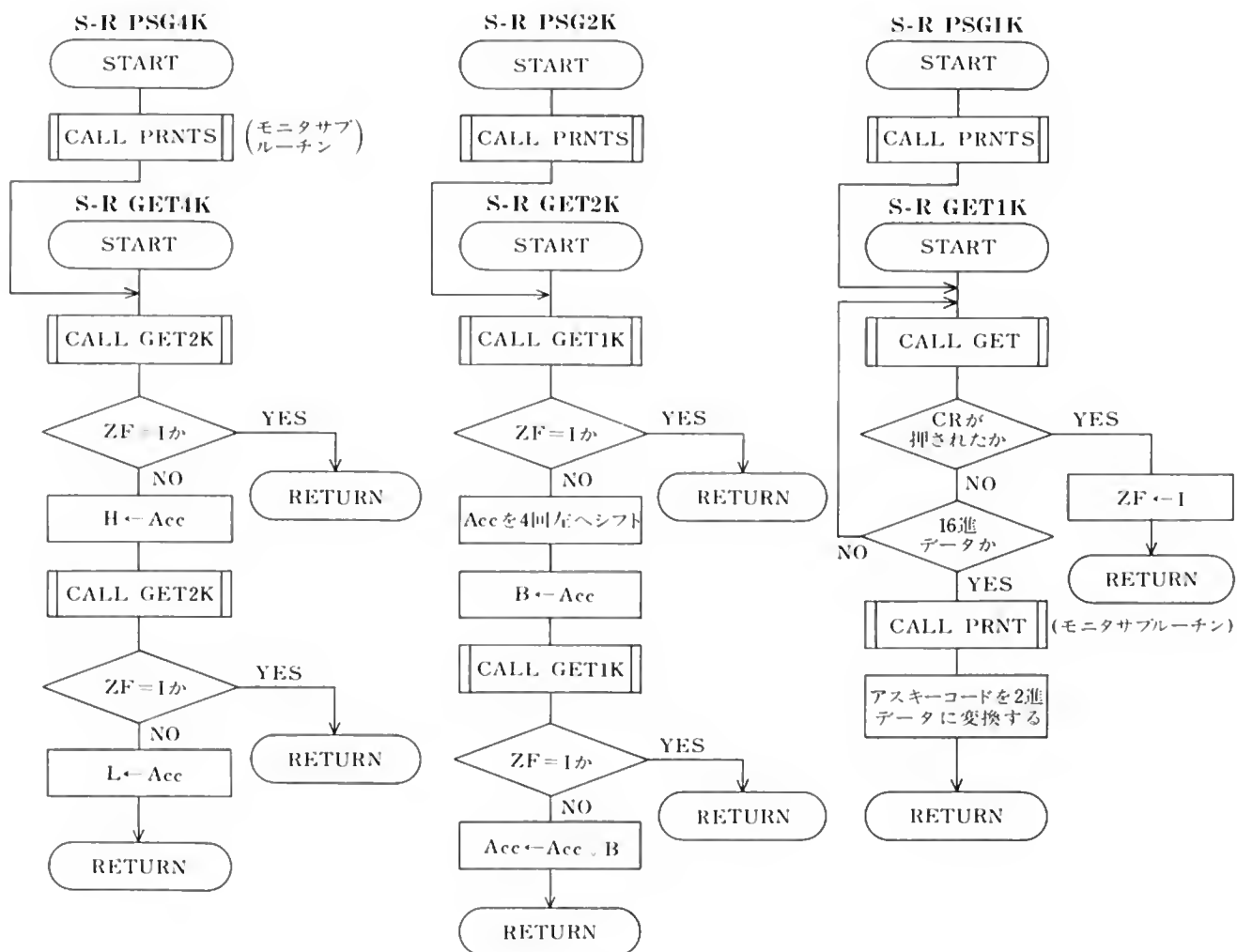
6

今度は、16進数でデータをキー入力するサブルーチンを作成する。データ入力の際に、カーソルを点滅させることにする。16進の入力桁は、1、2、4の各桁として、カーソルはその桁数だけキー入力されるまで点滅を続ける。16進数以外のキーが押されたら"ピッ"と音を立てる。キャリッジリターンが押されたら、Zフラグをセットしてリターンする。エコーバックおよび、チャタリングの防止も行なわれる。

サブルーチンコールは、スペースを置くか置かないかで各2通りの方法を作り合計次の6通りとする。

CALL GET4K (get 4hexa data)	HLの内容を16進4桁でキー入力
CALL PSG4K (print space, get 4hexa data)	スペースを表示しHLの内容を16進4桁でキー入力
CALL GET2K (get 2hexa data)	Accの内容を16進2桁でキー入力
CALL PSG2K (print space, get 2hexa data)	スペースを表示しAccの内容を16進2桁でキー入力
CALL GET1K (get 1hexa data)	Accの下位4ビットを16進1桁でキー入力
CALL PSG1K (print space, get 1hexa data)	スペースを表示しAccの下位4ビットを16進1桁でキー入力

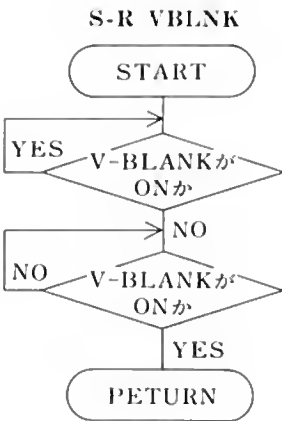
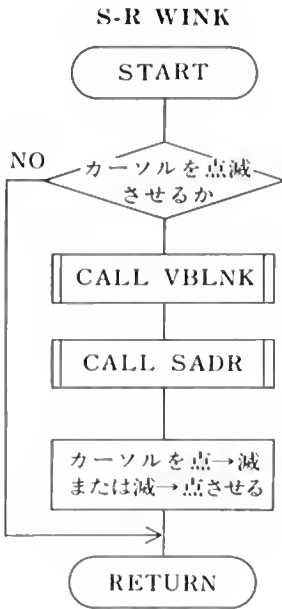
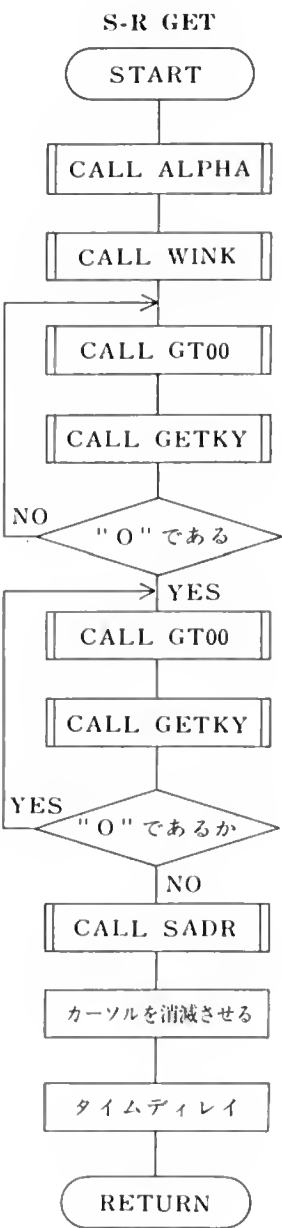
これらのサブルーチンも互いに関連があり、"GET4K"は"GET2K"を2回CALLし、"GET2K"は"GET1K"を2回CALLしている。さらに"GET1K"は、カーソル点滅およびモニタサブルーチンの"GETKY"の各ルーチンをCALLしているサブルーチン"GET"をCALLしている。まとめると次のようになる。



サブルーチン "GET" は、基本的に 2 つの要素からなっている。1 つは、カーソルを点滅させてキー入力待ちの状態を示すことであり、もう 1 つは、モニタサブルーチン "GETKY" を用いて実際に、キーコードを 1 つ Acc に取り込むことである。

カーソルの点滅、即ち、カーソルのディスプレイコードを V-RAM に転送するには、画面のチラつきを防止するために、V-Blank 信号 (E002 番地の D₇ が V-BLANK である) が ON の時に、コードの転送を行うようにしている。サブルーチン "VBLNK"。

キー入力される文字は、16進数データであるから、サブルーチン "ALPHA" によってキー取り込みモードを英数にしている。



右以外の他のサブルーチンの内容はプログラムを追って調べること。

プログラム中のエクスターナル "E" 表示のシンボルは、メモリマップド I-O に関するワークエリアのアドレスを示しており、詳細は、P.136 を参照せよ。

"SADR" は、V-RAM 中のカーソル位置を算出するサブルーチンであり、RETURN 時に、カーソル位置アドレスが HL レジスタに入る。

GET HEXA DATA SUB-ROUTINE

** Z80 ASSEMBLER SP-2101 PAGE 01 **

```

01 0000      :
02 0000      : GET 4  CHARACTER (DESTROYED:A,H,L)
03 0000      :      CALL GET4K
04 0000      :      CALL PSG4K
05 0000      :      EXIT:HL<--XXXX X:HEXA
06 0000      :
07 0000      PSG4K: ENT
08 0000 F5      PUSH  AF
09 0001 CD0000  E  CALL  PRNTS
10 0004 F1      POP   AF
11 0005      GET4K: ENT
12 0005 CD1500  CALL  GET2K
13 0008 C8      RET    Z
14 0009 67      LD     H, A
15 000A CD1500  CALL  GET2K
16 000D C8      RET    Z
17 000E 6F      LD     L, A
18 000F C9      RET
19 0010      SKP     2

```

"GET2K" を 2 回 CALL している。

```

22 0010      :
23 0010      : GET 2  CHARACTER (DESTROYED:A)
24 0010      :      CALL GET2K
25 0010      :      CALL PSG2K
26 0010      :      EXIT:A<--XX X:HEXA
27 0010      :
28 0010      PSG2K: ENT
29 0010 F5      PUSH  AF
30 0011 CD0000  E  CALL  PRNTS
31 0014 F1      POP   AF
32 0015      GET2K: ENT
33 0015 CD2B00  CALL  GET1K
34 0018 C8      RET    Z
35 0019 07      RLCA
36 001A 07      RLCA
37 001B 07      RLCA
38 001C 07      RLCA
39 001D C5      PUSH  BC
40 001E 47      LD     B, A
41 001F CD2B00  CALL  GET1K
42 0022 2802    JR     Z, + 4
43 0024 B0      OR     B
44 0025 04      INC    B
45 0026 C1      POP    BC
46 0027 C9      RET
47 0028      SKP     2

```

"GET1K" で Acc に 0X がセットされ、4 回左へローテートして X0 として Acc を B に入れておく。

もう一度 "GET1K" を行って Acc に 0X がセットされ、B と OR をとることによって Acc に XX がセットされる。INC B は Z フラグをリセットするためのもの。

```

50 0028      :
51 0028      : GET 1  CHARACTER (DESTROYED:A)
52 0028      :      CALL GET1K
53 0028      :      CALL PSG1K
54 0028      :      EXIT:A<--OX X:HEXA
55 0028      :
56 0028      PSG1K: ENT
57 0028 CD0000  E  CALL  PRNTS
58 002B      GET1K: ENT
59 002B CD5C00  CALL  GET
60 002E FE0D    CP     ODH

```

入力されたキーが **CR** であるか調べる。

** Z80 ASSEMBLER SP-2101 PAGE 02 **

01 0030 C8		RET	Z			<div>CR が押されたら</div>
02 0031 FE66		CP	66H			<div>Zフラグをリセッ</div>
03 0033 C8		RET	Z			<div>トして RETURN</div>
04 0034 F5		PUSH	AF			
05 0035 FE30		CP	30H			
06 0037 381D		JR	C, GGG2			
07 0039 FE3A		CP	3AH			
08 003B 3008		JR	NC, GGG0			
09 003D CD0000	E	CALL	PRNT			
10 0040 F1		POP	AF			
11 0041 D630		SUB	30H			
12 0043 180E		JR	GGG1			
13 0045 FE41	GGG0:	CP	41H			
14 0047 380D		JR	C, GGG2			
15 0049 FE47		CP	47H			
16 004B 3009		JR	NC, GGG2			
17 004D CD0000	E	CALL	PRNT			
18 0050 F1		POP	AF			
19 0051 D637		SUB	37H			
20 0053 FEFO	GGG1:	CP	FOH		:ZF<--0	
21 0055 C9		RET				<div>Zフラグをリセッ</div>
22 0056 F1	GGG2:	POP	AF		:ILLEGAL KEY HERE	<div>トして RETURN</div>
23 0057 CD0000	E	CALL	BELL			<div>16進データ以外の</div>
24 005A 18CF		JR	GET1K			<div>キーが押されたら</div>
25 005C		SKP	2			<div>BELL を鳴らす。</div>

28 005C	:				
29 005C	:	CURSOR WINK AND GETKEY			
30 005C	:	CALL	GET		
31 005C	:	EXIT:	A<--ASCII		
32 005C	:				
33 005C	GET:	ENT			
34 005C D5		PUSH	DE		
35 005D E5		PUSH	HL		
36 005E CD8800		CALL	ALPHA		
37 0061 CD9200		CALL	WINK		
38 0064 CD9E00		CALL	GTOO		
39 0067 CD0000	E	CALL	GETKY		
40 006A B7		OR	A		
41 006B 20F7		JR	NZ, -7		
42 006D CD9E00		CALL	GTOO		
43 0070 CD0000	E	CALL	GETKY		
44 0073 B7		OR	A		
45 0074 28F7		JR	Z, -7		
46 0076 CDCD00		CALL	SADR		
47 0079 3600		LD	(HL).OH	:DELETE CURSOR	
48 007B F5		PUSH	AF		
49 007C 112003		LD	DE, 000DH	:TIME DELAY	
50 007F 1B		DEC	DE		
51 0080 7A		LD	A, D		
52 0081 B3		OR	E		
53 0082 20FB		JR	NZ, -3		
54 0084 F1		POP	AF		
55 0085 E1		POP	HL		
56 0086 D1		POP	DE		
57 0087 C9		RET			
58 0088		SKP	H		

** Z80 ASSEMBLER SP-2101 PAGE 03 **

01 0088		ALPHA:	ENT		:FIX ALPHA MODE
02 0088	3E05		LD	A,05H	
03 008A	320000	E	LD	(KANST),A	
04 008D	AF		XOR	A	
05 008E	320000	E	LD	(KANAF),A	
06 0091	C9		RET		
07 0092			SKP	2	
10 0092		WINK:	ENT		
11 0092	AF		XOR	A	
12 0093	320000	E	LD	(KEYPA),A	
13 0096	21E500		LD	HL,WRKO	
14 0099	77		LD	(HL),A	
15 009A	2F		CPL		
16 009B	320000	E	LD	(KEYPA),A	
17 009E		GTOO:	ENT		
18 009E	3A0000	E	LD	A,(KEYPC)	
19 00A1	07		RLCA		
20 00A2	07		RLCA		
21 00A3	3813		JR	C,GTO2	
22 00A5	7E		LD	A,(HL)	
23 00A6	0F		RRCA		
24 00A7	D8		RET	C	
25 00A8	3EEF		LD	A,EFH	:A<--CURSOL PATTERN
26 00AA	CDBE00	GT01:	CALL	VBLNK	
27 00AD	EB		EX	DE,HL	
28 00AE	CDCD00		CALL	SADR	
29 00B1	77		LD	(HL),A	
30 00B2	EB		EX	DE,HL	
31 00B3	7E		LD	A,(HL)	:COMPLEMENT STATUS
32 00B4	EE01		XOR	01H	
33 00B6	77		LD	(HL),A	
34 00B7	C9		RET		
35 00B8	7E	GT02:	LD	A,(HL)	
36 00B9	0F		RRCA		
37 00BA	DO		RET	NC	
38 00BB	AF		XOR	A	:A<--SPACE PATTERN
39 00BC	18EC		JR	GT01	
40 00BE			SKP	2	
43 00BE		VBLNK:	ENT		
44 00BE	F5		PUSH	AF	
45 00BF	3A0000	E	LD	A,(KEYPC)	:V-BLANK CHECK
46 00C2	07		RLCA		
47 00C3	30FA		JR	NC,-4	
48 00C5	3A0000	E	LD	A,(KEYPC)	
49 00C8	07		RLCA		
50 00C9	38FA		JR	C,-4	
51 00CB	F1		POP	AF	
52 00CC	C9		RET		
53 00CD			SKP	H	

```

01 00CD          SADR:  ENT
02 00CD 2A0000  E      LD    HL,(DSPXY)          ;COMPUTE SCREEN ADR.
03 00D0  C5          PUSH  BC
04 00D1  D5          PUSH  DE
05 00D2  E5          PUSH  HL
06 00D3  C1          POP   BC
07 00D4  112800      LD    DE,28H
08 00D7  21D8CF      LD    HL,CFD8H
09 00DA  19          ADD   HL,DE
10 00DB  05          DEC   B
11 00DC  F2DA00      JP    P,-2
12 00DF  0600      LD    B,0H
13 00E1  09          ADD   HL,BC
14 00E2  D1          POP   DE
15 00E3  C1          POP   BC
16 00E4  C9          RET
17 00E5          SKP    2

20 00E5          WRKO:  ENT
21 00E5  00          DEFB  0H
22 00E6          END

```

このサブルーチンで用いられているモニタサブルーチンは、次のものがある。

PRNTS	(print space)	000CH
PRNT	(print accumulator)	0012H
BELL	(bell)	003EH
GETKY	(get key)	001BH

またメモリマップド I-O 及びモニタエリア内の

KEYPA	(key port A)	E000H
KEYPC	(key port C)	E002H
KANAF	(kana flag)	1170H
DSPXY	(display X-Y)	1171H

も用いられているので、これらのシンボルは、他のプログラムユニット中で定義されていなくてはならない。

〔応用〕 ここで作成したサブルーチンでは、16進データのセットが行われるが、キー入力の際りの対策として、DELETE 機能を付け加えることを考えよ。

たとえば、GET 4 Kを行うとき、

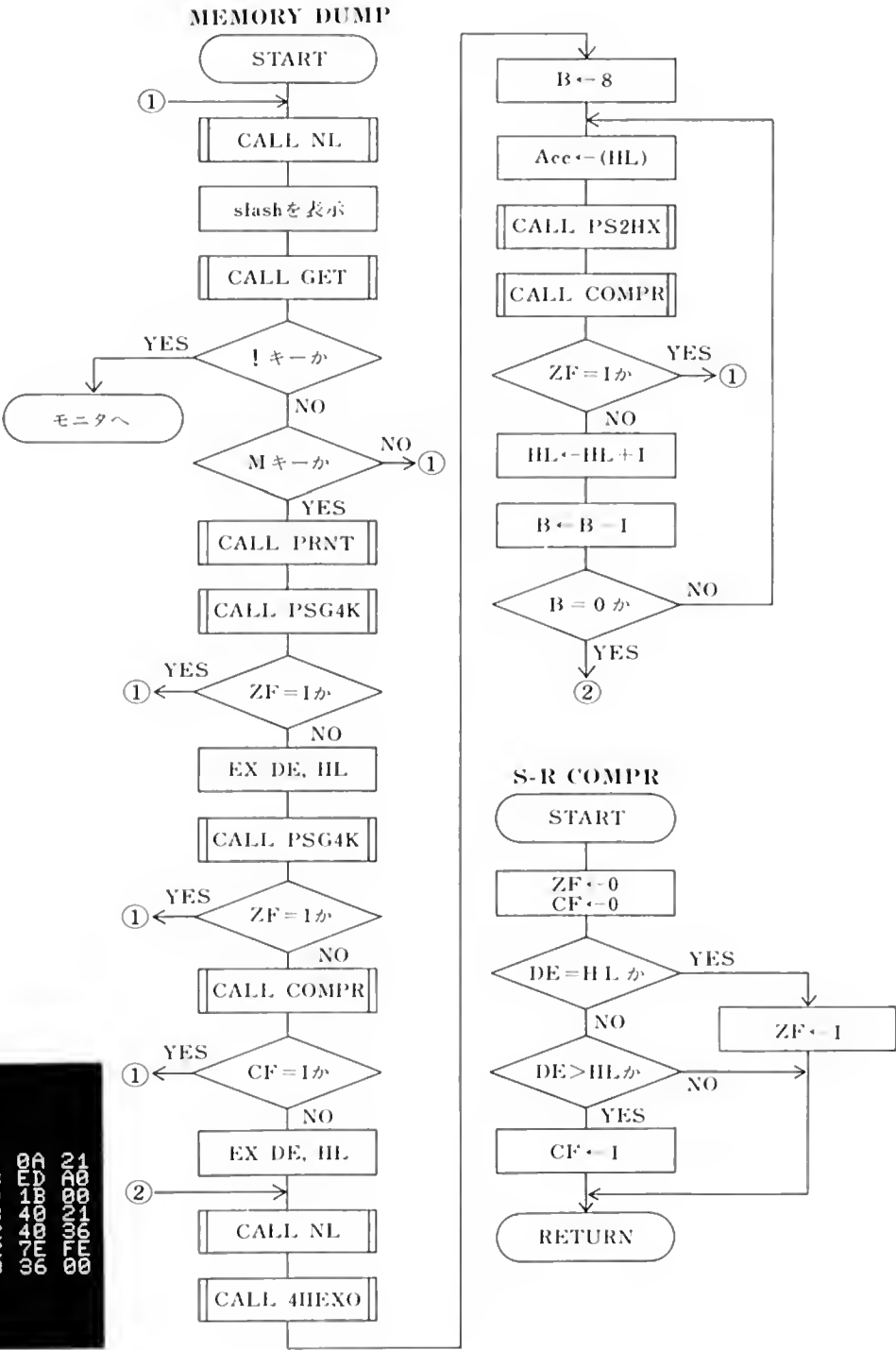
3A ❖

とキー入力して来て前の A を B に変更したい場合、DEL キーを押すとカーソルは 1 字戻り A が delete され、次に B を入力すればよいようにする。

7 前に作成した16進データのキー入力、表示のサブルーチンを用いて、デバッガの "M" コマンドの形式(或いはMACHINE LANGUAGEのMコマンドの形式)で、指定されたメモリブロックの内容を表示させるプログラムを作成する。

このプログラムは、初めにカーソルが点滅するコマンド待ちの状態となる。この時点でコマンドにはM、!があり、Mコマンドでスタート、!コマンドでモニタへジャンプするようにしておく。

Mコマンドでスタートしたら、メモリダンプするスタートアドレス(16進4桁)の入力待ちとなる。スタートアドレスが設定されたら、スペースを1個おいて、エンドアドレスの入力待ちとなる。エンドアドレスが指定されたら、メモリダンプを行い、終わったら再びコマンド待ちとなるようにする。



** Z80 ASSEMBLER SP-2101 PAGE 01 **

```

01 0000      ;
02 0000      ; MEMORY DUMP
03 0000      ; M:START
04 0000      ; !!GOTO MONITOR
05 0000      ;
06 0000      MEMRY: ENT
07 0000 CD0000 E CALL NL
08 0003 3E2F LD A,2FH
09 0005 CD0000 E CALL PRNT
10 0008 CD0000 E CALL GET
11 000B FE21 CP 21H
12 000D CA0000 E JP Z,MNTR
13 0010 FE4D CP 4DH
14 0012 2805 JR Z,+7
15 0014 CD0000 E MEMRO: CALL BELL
16 0017 18E7 JR MEMRY
17 0019 CD0000 E CALL PRNT
18 001C CD0000 E CALL PSG4K
19 001F 28F3 JR Z, MEMRO
20 0021 EB EX DE,HL
21 0022 CD0000 E CALL PSG4K
22 0025 28ED JR Z, MEMRO
23 0027 CD4300 CALL COMPR
24 002A 38E8 JR C, MEMRO
25 002C EB EX DE,HL
26 002D CD0000 E MEMR1: CALL NL
27 0030 CD0000 E CALL 4HEXO
28 0033 0608 LD B,8
29 0035 7E MEMR2: LD A,(HL)
30 0036 CD0000 E CALL PS2HX
31 0039 CD4300 CALL COMPR
32 003C 28C2 JR Z, MEMRO
33 003E 23 INC HL
34 003F 10F4 DJNZ MEMR2
35 0041 18EA JR MEMR1
36 0043 SKP H

```

改行して slash " / " を置きコマンド待ちとする。

" ! " が入力されるとモニタへ、
" M " が入力されるとスタートする。

メモリダンプするメモリブロックの
スタート、エンドアドレスの16進4
桁入力待ち。
スタートアドレスがエンドアドレス
よりも大だとキャンセルされる。

改行し、アドレスを表示する。

エンドアドレスまで1行に8バイト
ずつメモリダンプを行う。

COMPARE SUB-ROUTINE

** Z80 ASSEMBLER SP-2101 PAGE 02 **

```

01 0043      ;
02 0043      ; COMPARE DE,HL (DESTROYED:A)
03 0043      ; CALL COMPR
04 0043      ; EXIT:DE=HL Z=1
05 0043      ; DE>HL C=1
06 0043      ;
07 0043      COMPR: ENT
08 0043 7C LD A,H
09 0044 92 SUB D
10 0045 C0 RET NZ
11 0046 7D LD A,L
12 0047 93 SUB E
13 0048 C9 RET
14 0049      END

```

DE=HLの場合Zフラグがセットさ
れて RETURN。
DE>HLの場合Cフラグがセットさ
れて RETURN。

このプログラムでは、次の外部サブルーチン(モニタサブルーチンを除いて)が参照されている。

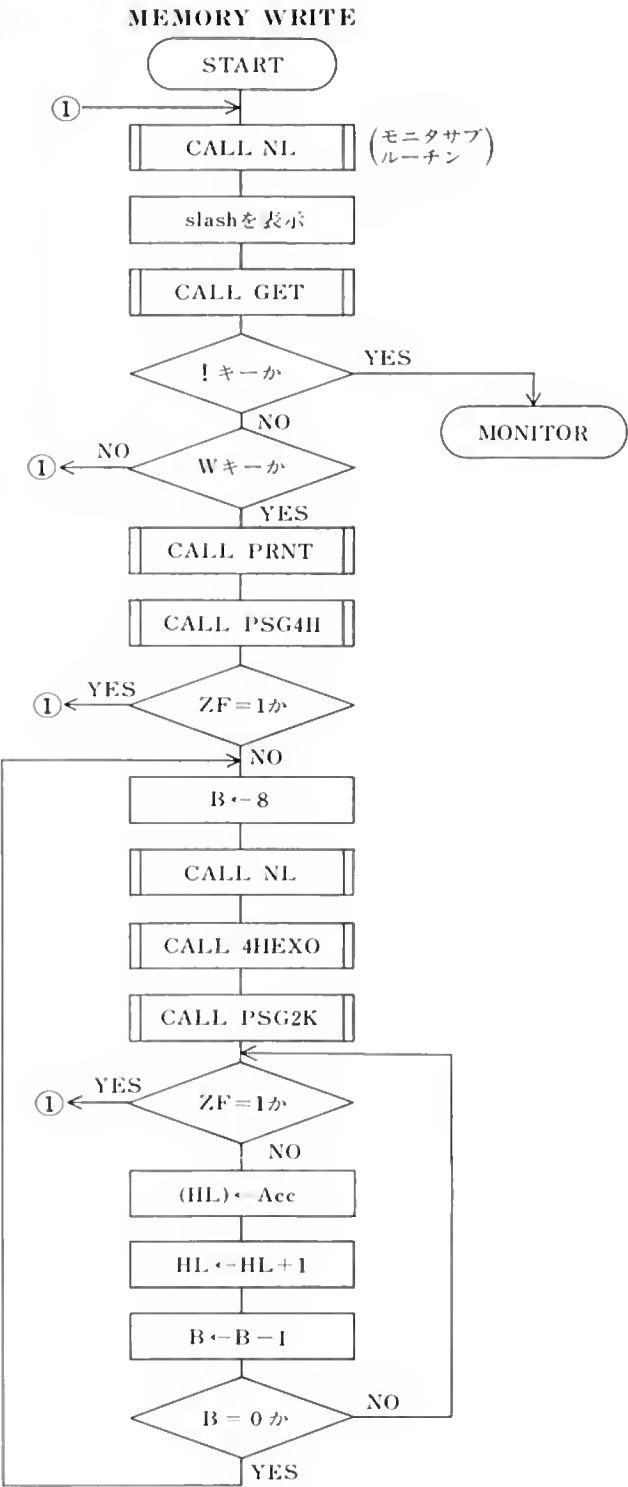
GET	(cursor wink and get key)	例題 6 参照
PSG4K	(print space, get 4 hexa data)	例題 6 参照
4HEXO	(4 hexa data out)	例題 5 参照
PS2HX	(print space, 2 hexa data out)	例題 5 参照

8 今度は、Wコマンドの形式で、メモリの指定アドレスから、16進2桁でデータを書き込むプログラムを作成する。

このプログラムは初めに、スラッシュ、カーソルのコマンド待ちの状態となり、Wコマンドでスタート、!コマンドでモニタへジャンプするようにしておく。

Wコマンドでスタートしたら、メモリライト (memory write) するスタートアドレス (16進4桁) の入力待ちとなる。スタートアドレスがGET4Kによってキー入力されたら、改行しアドレスを表示して、16進2桁のデータの入力待ちとなる。

Wコマンドを終えるには、**CR** キーを押す。



** Z80 ASSEMBLER SP-2101 PAGE 01 **

01 0000	:					
02 0000	:	MEMORY WRITE				
03 0000	:	W:START				
04 0000	:	!!GOTO MONITOR				
05 0000	:					
06 0000		WRITE:	ENT			
07 0000	CD0000	E	CALL	NL	改行してslash"/" (2FH)を置きコマンド待ちとする。	
08 0003	3E2F		LD	A,2FH		
09 0005	CD0000	E	CALL	PRNT		
10 0008	CD0000	E	CALL	GET		
11 000B	FE21		CP	21H	"! " (21H) が入力されるとモニターへ、" W " (57H) が入力されるとスタートする。	
12 000D	CA0000	E	JP	Z,MNTR		
13 0010	FE57		CP	57H		
14 0012	2805		JR	Z,+7		
15 0014	CD3E00		WRITEO:	CALL	BELL	
16 0017	18E7		JR	WRITE		
17 0019	CD0000	E	CALL	PRNT	Memory Write するスタートアドレスを待つ。	
18 001C	CD0000	E	CALL	PSG4K		
19 001F	28F3		JR	Z,WRITEO		
20 0021	0608		WRITE1:	LD	B,8	アドレスを表示し、1行に8バイトずつ表示しながら Memory Writeを行う。 Wコマンドの中止は [CR] キーの入力による。
21 0023	CD0000	E	CALL	NL		
22 0026	CD0000	E	CALL	4HEXO		
23 0029	CD0000	E	WRITE2:	CALL	PSG2K	
24 002C	28E6		JR	Z,WRITEO		
25 002E	77		LD	(HL),A		
26 002F	23		INC	HL		
27 0030	10F7		DJNZ	WRITE2		
28 0032	18ED		JR	WRITE1		
29 0034			SKP	H		

このプログラムでは、次の外部サブルーチンが参照されている。

GET	(cursor wink and get key)	例題 6 参照
PSG4K	(print space, get 4 hexa data)	例題 6 参照
4HEXO	(4 hexa data out)	例題 5 参照
PSG2K	(print space, get 2 hexa data)	例題 6 参照

また、" NL"," PRNT"などのモニターサブルーチンも、他のプログラムユニット中で定義されていなくてはならない。

〔応用〕 このWコマンドと、前のMコマンドと、更に実行コマンド、！コマンドを持つマシン語モニターを作成せよ。

実行コマンドGは、GET4Kで、スタートアドレスを指定して、プログラムを実行 (Program Counter:PCにスタートアドレスを設定すればよい) する。

/W	XXXX	memory write	
/M	XXXX	YYYY	memory dump
/G	XXXX	goto XXXX	……プログラムカウンタにXXXXをセットする。
/!		goto monitor	……0000番地にジャンプする。

プログラムは、MAINとSUB-ROUTINEの2つのプログラムユニットに分けて作成し、後でリンクさせるようにしてみよ。

9 指定されたメモリブロック内のアブソリュート形式プログラムを、1行に1命令ずつ、オペコードのアドレスと共にメモリダンプさせるプログラムを考える。

たとえば、3000から3009番地に、左のようなプログラムが入っているとして(バイナリで)、コマンドL(List)を行くと、右のようにオブジェクトリストが命令単位で表示されるようにしたい。

3000番地……CALL PRNT

XOR A

LD (HL), A

LD (1401H), A

JR +5

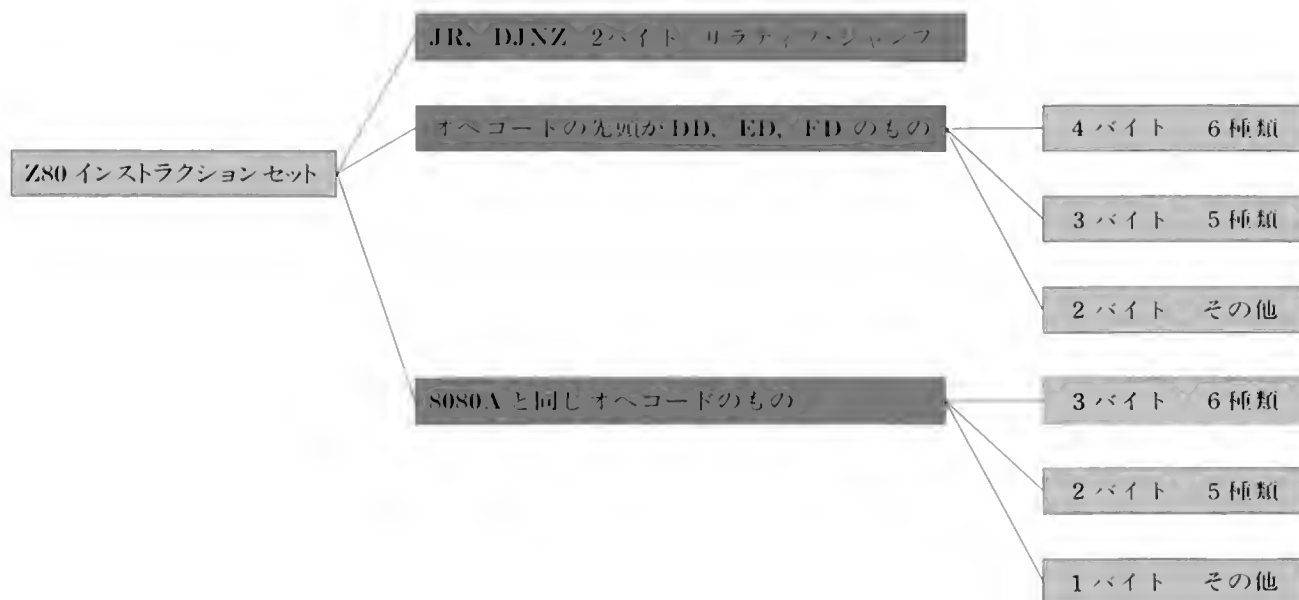
このバイナリが3000~3009
に入っているとする。



L コマンド	／L	3000	3009	
		3000	CD1200	3 バイト
		3003	AF	1 バイト
		3004	77	1 バイト
		3005	320114	3 バイト
		3008	1803	2 バイト
コマンド待ち	／❏			

このプログラムを作るには、まず、オペコードが、何バイト命令であるかの判別を行わなくてはならない。上の例では、オペコード"CD"が3バイト命令であり、"AF"が1バイト命令であるというように判別が行われなくてはならない。

Z80インストラクションセットには、8080ACPUのインストラクションセットの他に、リラティブ・ジャンプコマンド(JR, およびDJNZ)と、オペコードがDD, ED, FDで始まるコマンド群が加わっている。その構成は、次のようになっている。

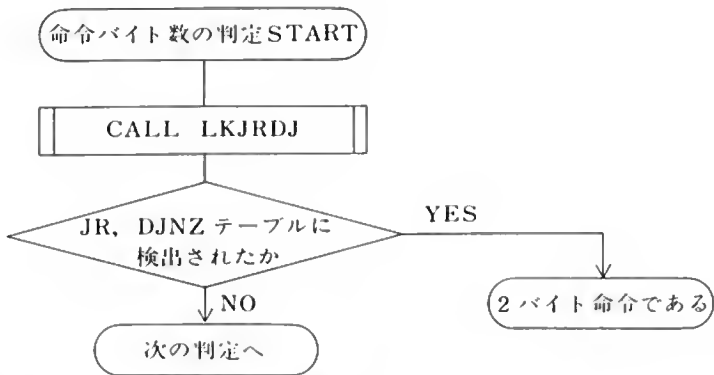


ここで、5種類、6種類というのは、オペレーションのバリエーション、たとえば、レジスタの違いなどの細分は考えず、オペコードをグループとして捉えた時に分けられる種類を示している。たとえば、次の8個の命令は1つのグループとして捉えることができる。すなわち、オペコード"00←r→110"を共通に持つ2バイト命令と考えることができる。オペコード中の"r"の3ビットが、7個のレジスタおよび(HL)の、合計8個のバリエーションを生むと考えられ、これを1種類とみるのである。

LD B,n LD C,n LD D,n LD E,n
LD H,n LD L,n LD A,n LD (HL),n

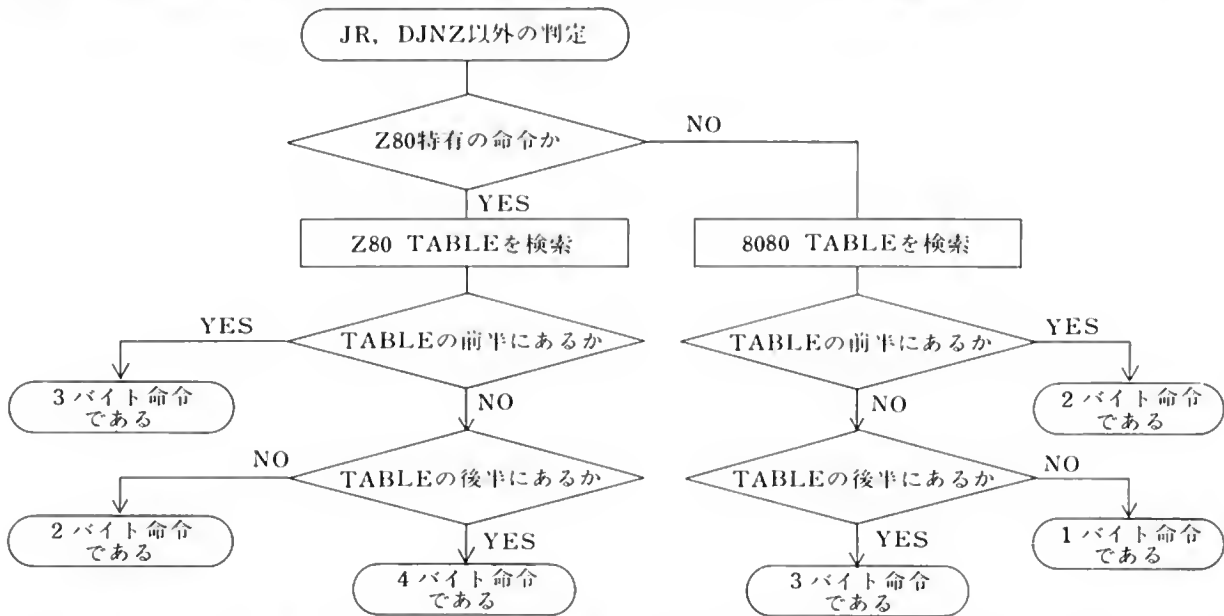
機械語の命令のバイト数を判別するには、はじめに、JR, DJNZの2バイトの相対的ジャンプ命令かどうかの判定を行う。すなわち、JR, DJNZの形のとるオペコードをテーブルに並べておいて、オペコードがそのテーブル中に検出されるかどうか調べる。

この検出は、プログラム中の"LKJRDJ" (LOOK JR, DJNZ TABLE) によって行われる。JR, DJNZ命令に含まれるオペコードは全部で6個であり、そのコードのテーブルを"JRDJT" (Z80 JR, DJNZ TABLE) に構成しておく。



JR, DJNZテーブルに検出されなかった場合は、前頁に示したように、8080ACPUと同じ命令か、Z80CPU特有の命令かの判定を行う。即ち、オペコードの先頭がDD, ED, FDであれば、Z80CPUに特有の命令であることになる。この判定によって参照するテーブルが分かれる。

前者、即ち8080Aと共通の命令である場合は"LOOK1" (LOOK UP 8080/Z80 TABLE) によって"8080T" (8080 TABLE) を検索し、後者の場合は、"LOOK1" によって"Z80TB" (Z80 TABLE) を検索する。TABLEの並びが同じ形で構成されているので、検索サブルーチン"LOOK1"は共通に使える。



TABLE検索によって上図のようにバイト数の分類が行われる。TABLEは、隣り合った2バイトずつが組みになっていて、はじめの1バイトが、ある命令型のグループ単位の共通部分、次の1バイトが、バリエーション（たとえばレジスタ、あるいはコンディションによる）のマスクを行うためのコードとなっている。

前頁に示した、オペコード"00←r→110"のグループ(8080のMVIコマンドグループ)は、8080TABLEの冒頭に置かれている。即ち、オペコードの共通部分、06Hが第1バイト目にあり、"←r→"の部分だけを0のビットとするためのレジスタマスクコードC7Hが第2バイト目に置かれている。調べるべきオペコードXXHと、06HのXORをとり、さらにその結果と、C7HとのANDをとった時、結果が00Hとなれば、オペコードXXHはこのグループに属することになる。


```

01 0000      :
02 0000      :   MEMORY LIST
03 0000      :   L:START
04 0000      :   !:GOTO MONITOR
05 0000      :
06 0000      LIST:   ENT
07 0000 CD0000   E   CALL   NL
08 0003 3E2F      LD   A,2FH
09 0005 CD0000   E   CALL   PRNT
10 0008 CD0000   E   CALL   GET
11 000B FE21      CP   21H
12 000D CA0000   E   JP    Z,MNTR
13 0010 FE4C      CP   4CH
14 0012 2805      JR    Z,+7
15 0014 CD0000   E LISTO: CALL   BELL
16 0017 18E7      JR    LIST
17 0019 CD0000   E   CALL   PRNT
18 001C CD0000   E   CALL   PSG4K
19 001F 28F3      JR    Z,LISTO
20 0021 EB        EX    DE,HL
21 0022 CD0000   E   CALL   PSG4K
22 0025 28ED      JR    Z,LISTO
23 0027 CD0000   E LIST1: CALL   GETKY
24 002A FE20      CP   20H
25 002C 2015      JR    NZ,LISTS
26 002E CD0000   E   CALL   GETKY
27 0031 B7        OR    A
28 0032 20FA      JR    NZ,4
29 0034 CD0000   E LISTP: CALL   GETKY
30 0037 FE0D      CP   0DH
31 0039 28D9      JR    Z,LISTO
32 003B FE66      CP   66H
33 003D 28D5      JR    Z,LISTO
34 003F FE20      CP   20H
35 0041 20F1      JR    NZ,LISTP
36 0043 CD0A01      LISTS: CALL   COMPR
37 0046 38CC      JR    C,LISTO
38 0048 CD0000   E   CALL   NL
39 004B ED530601   LD    (XXXX0),DE
40 004F 220801     LD    (YYYY0),HL
41 0052 EB        EX    DE,HL
42 0053 CD0000   E   CALL   4HEXO
43 0056 0601      LD    B,01H
44 0058 11FF00     LD    DE,JRDJT
45 005B CDB900     CALL   LKJRDJ
46 005E 2845      JR    Z,LIST4
47 0060 79        LD    A,C
48 0061 EB        EX    DE,HL
49 0062 FEDD      CP    DDH
50 0064 282D      JR    Z,LIST2
51 0066 FEED      CP    EDH
52 0068 2809      JR    Z,LIST55
53 006A FEFD      CP    FDH
54 006C 2825      JR    Z,LIST2
55 006E 21D500     LD    HL,8080T
56 0071 1825      JR    LIST3
57 0073 04        LIST55: INC    B
58 0074 13        INC    DE
59 0075 1A        LD    A,(DE)
60 0076 FE46      CP    46H
61 0078 CA9100     JP    Z,LIST66
62 007B FE56      CP    56H
63 007D CA9100     JP    Z,LIST66
64 0080 FE5E      CP    5EH
65 0082 CA9100     JP    Z,LIST66
66 0085 FE72      CP    72H

```

改行し, slash " / " を表示してコマンド待ちとする。
" L " が入力されるとスタートし,
" ! " が入力されるとモニタへ移る。

LISTING を行なうメモリブロックの入力待ち。[CR] が押されると戻る。

[SPACE] キーが押されたら LISTING を一旦停止する。
キーが離されるのを待つ。

[CR] が押されたらコマンド待ちへ戻る。

[SPACE] キーが押されたら LISTING を継続する。

BYTE カウンタを 1 にセットする。
JR, DJNZ 命令かを調べる。

DD, ED, FD で始まるコマンドであれば Z80TABLE を調べる。

違う時は, 8080A TABLE を調べる。

ED で始まり, 次が右のコードであれば, DD, FD の場合と異なり 2 バイト命令である。

```

01 0087 CA9100          JP      Z, LIST66
02 008A FE73           CP      73H
03 008C 2007          JR      NZ, LIST3-3
04 008E 0604          LD      B, 4
05 0090 B7            LIST66: OR      A
06 0091 1813          JR      LIST5
07 0093 04            LIST2: INC     B
08 0094 13            INC     DE
09 0095 21EA00         LD      HL, Z80TB
10 0098 CDC500         LIST3: CALL   LOOK1
11 009B FEFO          CP      FOH
12 009D 2807          JR      Z, LIST5
13 009F 79            LD      A, C
14 00A0 FE05          CP      05H
15 00A2 3801          JR      C, LIST4
16 00A4 04            INC     B
17 00A5 04            LIST4: INC     B
18 00A6 CD0000         E LIST5: CALL   PRNTS
19 00A9 ED5B0601       LD      DE, (XXXX0)
20 00AD 1A            LIST6: LD      A, (DE)
21 00AE CD0000         E      CALL   2HEXO
22 00B1 13            INC     DE
23 00B2 10F9          DJNZ    LIST6
24 00B4 2A0801        LD      HL, (YYYY0)
25 00B7 188E          JR      LIST1
26 00B9              :
27 00B9              : LOOK JR DJNZ TABLE
28 00B9              :
29 00B9              LKJRDJ: ENT
30 00B9 4E            LD      C, (HL)
31 00BA 1B            DEC     DE
32 00BB 13            LKJDO: INC     DE
33 00BC 1A            LD      A, (DE)
34 00BD B9            CP      C
35 00BE C8            RET     Z
36 00BF FEFO          CP      FOH
37 00C1 20F8          JR      NZ, LKJDO
38 00C3 B7            OR      A
39 00C4 C9            RET
40 00C5              :
41 00C5              : LOOK UP 8080/Z80 TABLE
42 00C5              : (DESTROYED: A, C, H, L)
43 00C5              : IN: HL<--TOP ADR OF TBL
44 00C5              : DE<--OPECODE ADR
45 00C5              : CALL LOOK1
46 00C5              : EXT: A=F0 TABLE END
47 00C5              : C=DATA# OF TBL
48 00C5              :
49 00C5              LOOK1: ENT
50 00C5 0E00          LD      C, OH
51 00C7 2B            DEC     HL
52 00C8 23            LK00: INC     HL
53 00C9 0C            INC     C
54 00CA 7E            LD      A, (HL)
55 00CB FEFO          CP      FOH
56 00CD C8            RET     Z
57 00CE 1A            LD      A, (DE)
58 00CF AE            XOR     (HL)
59 00D0 23            INC     HL
60 00D1 A6            AND     (HL)
61 00D2 20F4          JR      NZ, LK00
62 00D4 C9            RET

```

ED73で始まる命令は4バイト命令である。

DD, ED, FDで始るZ80固有の命令では、バイトカウンタに1を足しておき、さらにオペコードとして意味のある第2バイト目のアドレスをDEにセットしておく。

テーブルに無い場合バイトカウンタはそのまま

テーブルの前半にあればB←B+1
後半にあればB←B+2

スペースを1個表示して、命令のバイト数だけコードの表示(2HEXOを使う)を行う。

オペコードをCに入れる。

JR, DJNZのテーブルにオペコードが見つかったらZF=1の状態です。
TURN。

見つからない時はZF=0でRETURN
(AccはFOHだから"OR A"でZFはリセットされる)

Cレジスタに00をセット。

バイト数を調べようとするオペコードとオペコードのパターンとのXOR(排他的論理和)を行ない、次にレジスタまたはコンディションのマスクパターンとANDをとった時ゼロになれば、オペコードはそのグループに属することになる。テーブルの何番目に見つかったかは、Cレジスタに残る。見つからないと、AccはF0となる。

** Z80 ASSEMBLER SP-2101 PAGE 03 **

```

01 00D5          ;
02 00D5          ; 8080 TABLE
03 00D5          ;
04 00D5          8080T:  ENT                      ;8080A BYTE SIZE
05 00D5 06       DEFB  06H                      ;MVI(2 BYTE)
06 00D6 C7       DEFB  C7H                      ;REGISTER MASK
07 00D7 C6       DEFB  C6H                      ;DIRECT ALU
08 00D8 C7       DEFB  C7H                      ;OPERATION MASK
09 00D9 DB       DEFB  DBH                      ;IN,OUT
10 00DA F7       DEFB  F7H                      ;MASK
11 00DB CB       DEFB  CBH                      ;RLC, RRC, BIT, SET
12 00DC FF       DEFB  FFH                      ;UNCONDITIONAL
13 00DD          ;
14 00DD          ;
15 00DD 01       DEFB  01H                      ;LXI(3 BYTE)
16 00DE CF       DEFB  CFH                      ;REGISTER MASK
17 00DF 22       DEFB  22H                      ;STA, LDA, LHLD, SHLD
18 00E0 E7       DEFB  E7H                      ;MASK
19 00E1 C2       DEFB  C2H                      ;JXX
20 00E2 C7       DEFB  C7H                      ;CONDITION MASK
21 00E3 C4       DEFB  C4H                      ;CXX
22 00E4 C7       DEFB  C7H                      ;CONDITION MASK
23 00E5 C3       DEFB  C3H                      ;JMP
24 00E6 FF       DEFB  FFH                      ;UNCONDITIONAL
25 00E7 CD       DEFB  CDH                      ;CALL
26 00E8 FF       DEFB  FFH                      ;UNCONDITIONAL
27 00E9 FO       DEFB  FOH                      ;TABLE END
28 00EA          SKP  2

```

```

31 00EA          ;
32 00EA          ; Z80 TABLE
33 00EA          ;
34 00EA          Z80TB:  ENT                      ;Z80 BYTE SIZE
35 00EA 46       DEFB  46H                      ;(3 BYTE)
36 00EB C7       DEFB  C7H
37 00EC 70       DEFB  70H
38 00ED F8       DEFB  F8H
39 00EE 86       DEFB  86H
40 00EF C7       DEFB  C7H
41 00F0 34       DEFB  34H
42 00F1 FE       DEFB  FEH
43 00F2          ;
44 00F2          ;
45 00F2 36       DEFB  36H                      ;(4 BYTE)
46 00F3 FF       DEFB  FFH
47 00F4 21       DEFB  21H
48 00F5 FF       DEFB  FFH
49 00F6 2A       DEFB  2AH
50 00F7 FF       DEFB  FFH
51 00F8 22       DEFB  22H
52 00F9 FF       DEFB  FFH
53 00FA CB       DEFB  CBH
54 00FB FF       DEFB  FFH
55 00FC 43       DEFB  43H
56 00FD C7       DEFB  C7H
57 00FE FO       DEFB  FOH                      ;TABLE END
58 00FF          SKP  H

```

8080A 型命令の TABLE および、DD、ED、FD ではじまる Z80 固有の命令の第2バイト目のオペコード判別のための TABLE が示されている。

両者には対照性があり、前半の5種類はバイトカウンタに1を足すもの、後半の6種類はバイトカウンタに2を足すものであり、TABLE に見つからない時はバイトカウンタはそのままにする。ただし、バイトカウンタは、8080A の TABLE を参照する時は0、Z80の TABLE を参照する時は1になっていなくてはならない。

** Z80 ASSEMBLER SP-2101 PAGE 04 **

```

01 00FF      ;
02 00FF      ; Z80 JR,DJNZ TABLE
03 00FF      ;
04 00FF      JRDJT: ENT
05 00FF 10      DEFB 10H
06 0100 18      DEFB 18H
07 0101 20      DEFB 20H
08 0102 28      DEFB 28H
09 0103 30      DEFB 30H
10 0104 38      DEFB 38H
11 0105 FO      DEFB FOH
12 0106      SKP 2

15 0106      ;
16 0106      ;
17 0106      XXXX0: ENT
18 0106      DEFS 2
19 0108      YYYY0: ENT
20 0108      DEFS 2
21 010A      SKP 2

24 010A      ;
25 010A      ; COMPARE DE,HL
26 010A      ;
27 010A      COMPR: ENT
28 010A      LD A,H
29 010B      SUB D
30 010C      RET NZ
31 010D      LD A,L
32 010E      SUB E
33 010F      RET

```

(2 BYTE)
相対ジャンプ JR, DJNZ のテーブル。

オペコードのアドレスと、LIST を
行うブロックのエンドアドレスを入
れるためのバッファ。

DE と HL の比較サブルーチン。
DE=HL の時 ZF=1
DE>HL の時 CF=1

このプログラムの中で用いられているサブルーチンは次のものである。

GET	(cursor wink and get key)	例題 6 参照
PSG4K	(print space, get 4 hexa)	例題 6 参照
4HEX0	(4 hexa deta out)	例題 5 参照
2HEX0	(2 hexa data out)	例題 5 参照

なおモニタサブルーチン等の定数に関する EQU の部分はこのリスト上には省かれている。

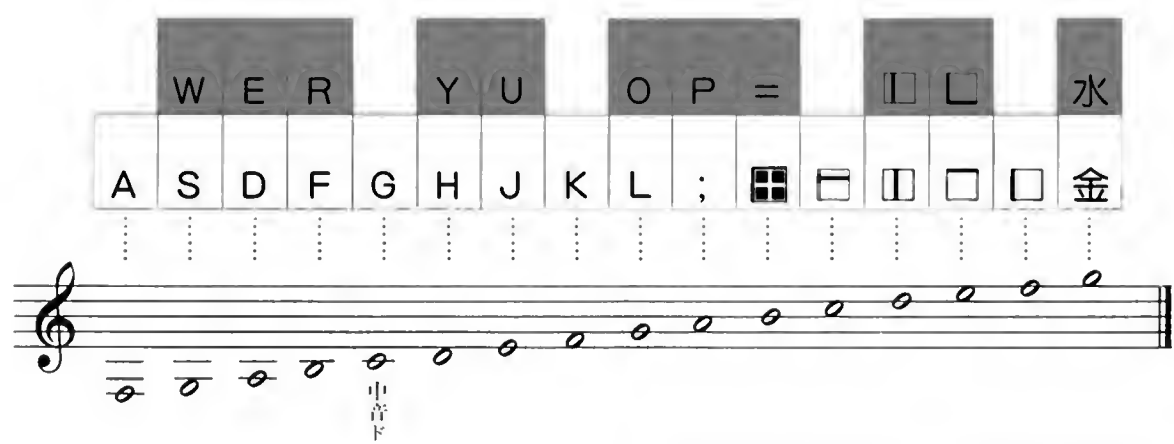
〔応用〕 M コマンド、W コマンド、L コマンド、G コマンド、(! コマンド) を持つマシン語モニタを作成せよ。G コマンドでステップ動作をさせるには、どうすればよいか考えよ。

〔応用〕 Z80 マシン語オペコードの分類を進めて、ニモニクのテーブルを作成し、Z80 DIS ASSEMBLER (逆アセンブラ) を作成せよ。

10 キーボードを鍵盤とする電子オルガンを作成する。手前から3段目のキー（左端の " A " から右端の " 金 " ま）を白鍵として、4段目のキーを黒鍵として使い、中音のハ調のオクターブを挟んで、上下に五度ずつの音域をとることにしよう。

キーの取り込みは、モニタサブルーチンの " GETKY " を使用し、音を出したり止めたりするのは、モニタサブルーチンの " MSTA "、" MSTP " を使用する。

MZ = 80K のキーボードと、音階との関係を下図に示す。図はキーをわざと縦長に描いてあり、さらに、黒鍵として使うキーを黒くしてみた。



" GETKY " によってキーコードをとり込んだら、所定の音を作らなくてはならない。

モニタサブルーチン " MSTA " をコールした時に鳴る音は、2 MHz を、11A 1、11A 2 番地の 2 バイトデータで分周した周波数の音であるから、所望の音の周波数に近い音を得るような分周比をあらかじめ計算しておくてはならない。

発生周波数 $f(\text{Hz}) = 2(\text{MHz}) / \text{分周比}$

右の表は上の式に従って、それぞれの周波数を得るために分周比を算出したものである。

表中のカッコで表示してある周波数が音名に対しての平均率音階であるが、本プログラムでは実測データより分周比を算出した。

音 名	周波数 (Hz)	分 周 比
低音ファ	175(174.6)	2CA4
#ファ	186(185)	2A00
ソ	197(196)	27A8
#ソ	208(207.5)	2582
ラ	222(220)	2331
#ラ	233(233.1)	2187
シ	245(246.9)	1FE3
中音ド	261(261.6)	1DEF
#ド	277(277.2)	1C34
レ	294(293.7)	1A92
#レ	311(311.1)	191E
ミ	329(329.6)	17BF
ファ	350(349.2)	1652
#ファ	373(370)	14F1
ソ	394(392)	13D4
#ソ	417(415)	12BC
ラ	444(440)	1198
#ラ	466(466.2)	10C3
シ	490(493.2)	0FF1
高音ド	522(523.3)	0EF7
#ド	553(554.4)	0E20
レ	590(587.3)	0D3D
#レ	621(622.3)	0C94
ミ	658(659.3)	0BDF
ファ	699(698.5)	0B2D
#ファ	745(740)	0A7C
ソ	788(784)	09EA

プログラムのフローチャートを右に示す。"GETKY"によって、Accにキーのアスキーコードがとり込まれる。有効なキーは、鍵盤となるキーと、モニターへ制御を移すための、"! "キーとする。

!GOTO MONITOR

鍵盤のキーが押されたなら、出るべき音の分周比を11A1, 11A2番地にストアして、"MSTA"をコールする。この場合、GETKYされたキーコードと分周比を表にしたテーブルを作成しておく。

この音階のテーブル (SCALE TABLE) は、キーコードと、分周比の合計3バイトずつを並べて、次のように構成する。

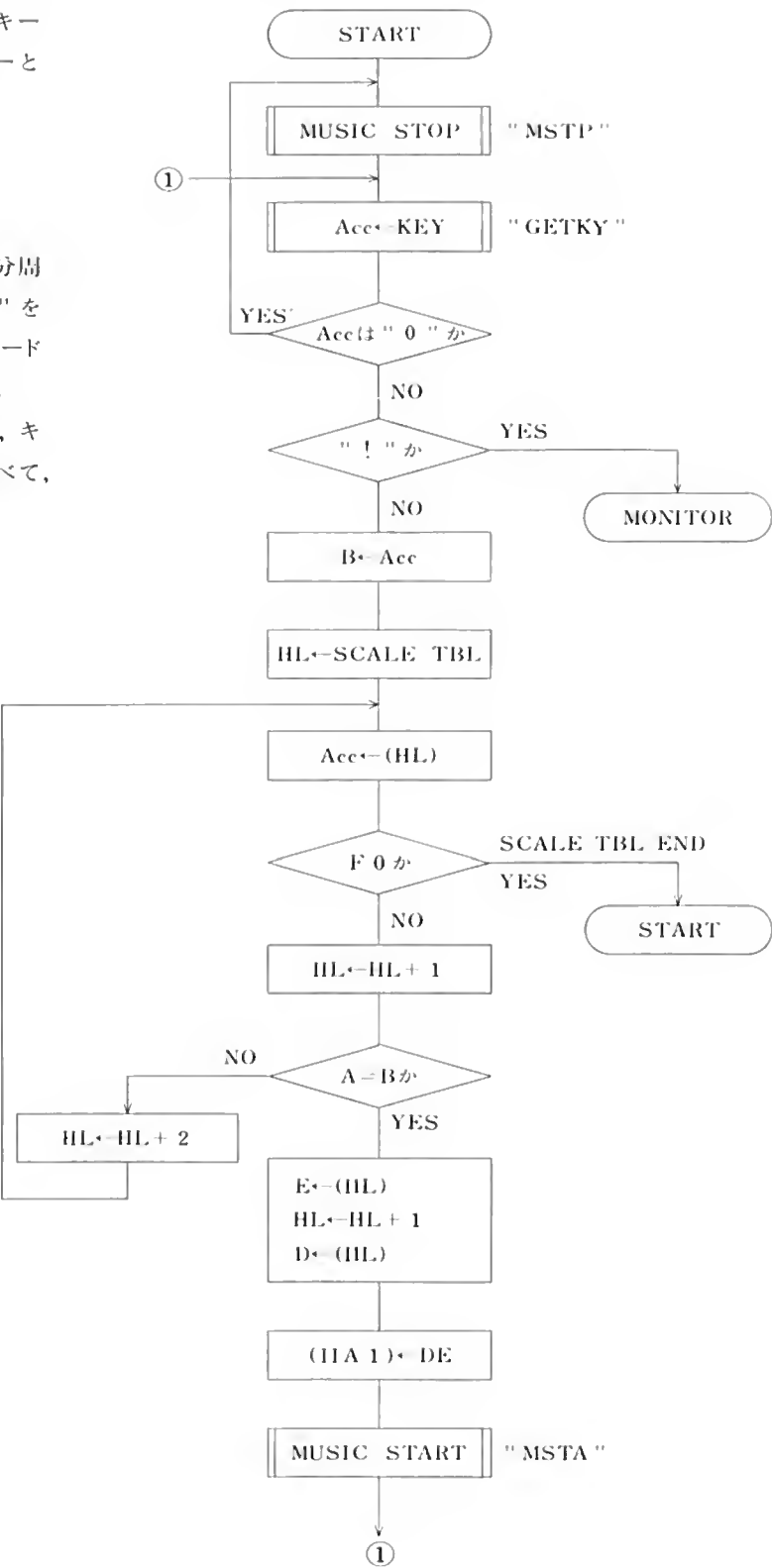
SCALE TABL

KEY
分 周 比
KEY
分 周 比
...

SCALE TABLE中にKEYコードを見つけたら次の2バイトを分周比として、11A1, 11A2番地にストアするようにする。

SCALE TABLE中にKEYが見つからなかった場合、(TABLE ENDは、F0とする) STARTへ戻って音を止める。

FLOWCHART



電子オルガンプログラムのアセンブリリストを次に示す。

```

** Z80 ASSEMBLER SP-2101 PAGE 01 **

01 0000 P      MNTR: EQU 0000H
02 0000 P      MSTP: EQU 0047H
03 0000 P      MSTA: EQU 0044H
04 0000 P      GETKY: EQU 001BH
05 0000        SKP 3

                                ] モニタサブルーチンをラベルシンボルに
                                ] 定義する

09 0000 CD4700  STARTO: CALL MSTP
10 0003 CD1B00  START1: CALL GETKY
11 0006 B7      OR A
12 0007 28F7    JR Z, STARTO
13 0009 FE21    CP 21H
14 000B CA0000  JP Z, MNTR
15 000E 47      LD B, A
16 000F 212B00  LD HL, SCTBL
17 0012 7E      CMPR: LD A, (HL)
18 0013 FEFO    CP FOH
19 0015 28E9    JR Z, STARTO
20 0017 23      INC HL
21 0018 B8      CP B
22 0019 2804    JR Z, +6
23 001B 23      INC HL
24 001C 23      INC HL
25 001D 18F3    JR CMPR
26 001F 5E      LD E, (HL)
27 0020 23      INC HL
28 0021 56      LD D, (HL)
29 0022 ED53A111 LD (11A1H), DE
30 0026 CD4400  CALL MSTA
31 0029 18D8    JR START1
32 002B        SKP 3

                                ] 11A1, 11A2番地に分周比をストアして
                                ] "MSTA"をコール

36 002B 41      SCTBL: DEFB 41H ; "A" KEY
37 002C A42C    DEFW 2CA4H ; □ファ
38 002E 57      DEFB 57H ; "W" KEY
39 002F 002A    DEFW 2A00H ; □#ファ
40 0031 53      DEFB 53H ; "S" KEY
41 0032 A827    DEFW 27A8H ; □ソ
42 0034 45      DEFB 45H ; "E" KEY
43 0035 8225    DEFW 2582H ; □#ソ
44 0037 44      DEFB 44H ; "D" KEY
45 0038 3123    DEFW 2331H ; □ラ
46 003A 52      DEFB 52H ; "R" KEY
47 003B 8721    DEFW 2187H ; □#ラ
48 003D 46      DEFB 46H ; "F" KEY
49 003E E31F    DEFW 1FE3H ; □シ
50 0040 47      DEFB 47H ; "G" KEY
51 0041 EE1D    DEFW 1DEEH ; フ
52 0043 59      DEFB 59H ; "Y" KEY
53 0044 341C    DEFW 1C34H ; #フ
54 0046 48      DEFB 48H ; "H" KEY
55 0047 921A    DEFW 1A92H ; レ
56 0049 55      DEFB 55H ; "U" KEY
57 004A 1E19    DEFW 191EH ; #レ
58 004C 4A      DEFB 4AH ; "J" KEY
59 004D BF17    DEFW 17BFH ; ミ
60 004F 4B      DEFB 4BH ; "K" KEY

                                SCALE TABLE の先頭

```

01	0050	5216	DEFW	1652H	;ファ
02	0052	4F	DEFB	4FH	; "O" KEY
03	0053	F114	DEFW	14F1H	; #ファ
04	0055	4C	DEFB	4CH	; "L" KEY
05	0056	D413	DEFW	13D4H	;ソ
06	0058	50	DEFB	50H	; "P" KEY
07	0059	BC12	DEFW	12BCH	; #ソ
08	005B	3B	DEFB	3BH	; " ; " KEY
09	005C	9811	DEFW	1198H	;ラ
10	005E	3D	DEFB	3DH	; " = " KEY
11	005F	C310	DEFW	10C3H	; #ラ
12	0061	FB	DEFB	FBH	; " + " KEY
13	0062	F10F	DEFW	0FF1H	;シ
14	0064	E3	DEFB	E3H	; " □ " KEY
15	0065	F70E	DEFW	0EF7H	; □ ド
16	0067	F4	DEFB	F4H	; " □ " KEY
17	0068	200E	DEFW	0E20H	; □ # ド
18	006A	E2	DEFB	E2H	; " □ " KEY
19	006B	3D0D	DEFW	0D3DH	; □ レ
20	006D	EC	DEFB	ECH	; " □ " KEY
21	006E	940C	DEFW	0C94H	; □ # レ
22	0070	D7	DEFB	D7H	; " □ " KEY
23	0071	DF0B	DEFW	0BDFH	; □ ミ
24	0073	D4	DEFB	D4H	; " □ " KEY
25	0074	2D0B	DEFW	0B2DH	; □ ファ
26	0076	73	DEFB	73H	; " 水 " KEY
27	0077	7C0A	DEFW	0A7CH	; □ # ファ
28	0079	75	DEFB	75H	; " 金 " KEY
29	007A	EA09	DEFW	09EAH	; □ ソ
30	007C	FO	DEFB	FOH	; SCALE TBL END
31	007D		END		

〔応用〕 アルペッジオを出す伴奏用オルガンを作成せよ。

たとえば、Cを押した時にはCのコードのアルペッジオが鳴り、Amollを出すときは何かキーを指定して行くというように。

アルペッジオの形の例を右に示す。ギターの代りに使うことができるかも知れない。

〔応用〕 今回作成したオルガンに、繰り返し演奏機能を付け加えよ。つまり、キーを押して演奏した通りに自動演奏をさせる。



Cのアルペッジオ

Amollのアルペッジオ

この場合、押されたキーのコードを記憶するだけでなく、押された時間（音長）も記憶していなくてはならないから、タイムカウンタを考える必要がある。モニタサブルーチン"TIMST" "TIMRD"は、秒単位しかカウントできないので、適当な、タイムカウトルーチンを作成するか、E002番地のTEMPO用のカウントデータを用いるとよい。

E002番地の内容は、CPUボード上の発振器555(IC1)により決まる周波数で、0,1をくり返している。

(P.136を参照)

第 7 章

MZ-80K システムコントロール 資料と解説

117-80K



7-1 アスキーコード表

アスキーコードと、モニタサブルーチンとの関係は、次項「モニタサブルーチンの使い方」で解説される。
表の見方は、上位4ビットがコラムに相当し、下位4ビットが、行に相当しており、たとえば、キャラクタ
"A"のアスキーコードは、16進コードで41Hであり、"Z"は5AHである。

ただし、表中の11H~16Hまでのコードは、カーソルコントロールのためのアスキーコードである。たとえばACCの内容が15Hの時、CALL PRNT(モニタサブルーチン)を行うとカーソルホームが行われる。("H"を表示するのではない)

カッコのついていないカナ文字、たとえば"ァ" (87H)、などは小文字を表わす。

ASCII

MSD LSD		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000			SP	0	@	P	☛	日	SP	一	夕	≡	SP	☐	☐	☐
1	0001		↓	!	1	A	Q	H	月	。	ア	チ	厶	▮	☐	♠	●
2	0010		↑	"	2	B	R	工	火	「	イ	ツ	メ	▮	☐	▮	▮
3	0011		→	#	3	C	S	大	水	」	ウ	テ	モ	☐	☐	☐	♥
4	0100		←	\$	4	D	T	+	木	、	エ	ト	ヤ	☐	☐	☐	▮
5	0101		H	%	5	E	U	✱	金	・	オ	ナ	ユ	☐	▮	☐	▮
6	0110		C	&	6	F	V	¥	土	ヲ	カ	ニ	ヨ	☐	▮	☐	☒
7	0111			!	7	G	W	☺	生	ア	ギ	ヌ	ラ	☐	☐	▮	☐
8	1000			(8	H	X	☹	年	イ	ク	ネ	リ	☐	☐	▮	♣
9	1001)	9	I	Y	☹	時	ウ	ケ	ノ	ル	▮	☐	▮	▮
A	1010			*	:	J	Z	✱	分	エ	コ	ハ	レ	☐	☐	☐	♦
B	1011			+	;	K	[✱	秒	オ	サ	ヒ	☐	☐	☐	☐	☐
C	1100			,	<	L	\	☐	円	ヤ	シ	フ	☐	☐	☐	☐	☐
D	1101	CR		一	≡	M]	☐	¥	ユ	ス	ヘ	ン	☐	☐	▮	▮
E	1110			□	>	N	↑	☐	£	ヨ	セ	ホ	〃	☐	☐	▮	▮
F	1111			/	?	O	←	☐		ツ	ソ	マ	。	☐	☐	☐	π

7-2 モニタサブルーチンの使い方

SP-1002のモニタサブルーチンには次のものがある。ここで、用いているサブルーチン名は、サブルーチンの機能をシンボリックに表現したものである。

各サブルーチンで取り扱われるコードについては特に注意が必要である。表中で述べられるコードはいずれも16進コード表現である。

サブルーチン名 (16進番地)	サブルーチンの機能	レジスタ 保 存	スタック 数
CALL LETNL (0006)	行を替えて、次の行の先頭にカーソルをセットする。	AF 以外は 保存	8
CALL PRNTS (000C)	テレビ画面のカーソル位置にスペースを1個だけ表示する。	AF 以外は 保存	13
CALL PRNT (0012)	ACCにあるデータをASCiiコードと見て、テレビ画面のカーソル位置に表示する。ASCiiコードと、キャラクタの関係は、前頁の表を参照。ただし、0Dコードの時は、キャリッジリターンが実行され、11～16コードの時は、それぞれカーソルコントロールが実行される。	AF 以外は 保存	13
CALL MSG (0015)	テレビ画面のカーソル位置から、メッセージを表示する。メッセージの先頭アドレスは、あらかじめDEレジスタに指定しておくこと。メッセージは、ASCiiコードで構成し、エンドマークは、0Dコードでなくてはならない。ただし、キャリッジリターンは実行されない。カーソルコントロール(11～16コード)は実行される。	全レジスタ 保存	13
CALL BELL (003E)	中音のラ(約440Hz)を瞬間だけ鳴らす。	AF 以外は 保存	5
CALL MELDY (0030)	音楽データを演奏する。音楽データの先頭アドレスは、あらかじめ、DEレジスタに指定しておくこと。音楽データは、「BASIC解説書」88～89ページに示したストリングと同様に、音程、音長の順にASCiiコードで表現し、エンドマークは、0DまたはC8(キャラクタは■)でなければならない。但し、リターン時にCフラグが0なら演奏完了、Cフラグが1なら途中で[BREAK]キーが押されたことを示す。	AF 以外は 保存	7
CALL XTEMP (0041)	演奏テンポを設定する。テンポのデータ(01から07)をACCにセットしてコールする。 ACC←01 最も遅い ACC←04 中位の速度 ACC←07 最も速い 注意が必要なのは、ACCに入れるコードは、1から7までのバイナリコードであり、1から7に相当するASCiiコード(31から37)ではない点である。	全レジスタ 保存	4
CALL MSTA (0044)	指定された分周比の音を連続して鳴らす。 分周比nn'(2バイトのデータ)は、11A1番地にn', 11A2番地にnをストアしてコールする。 分周比と発生周波数の関係は $2\text{MHz}/nn'$	BC, DE だけ 保存	3

サブルーチン名 (16進番地)	サブルーチンの機能	レジスタ 保 存	スタック 数
CALL MSTP (0047)	音の発生をとめる。	AF 以外は 保存	1
CALL TIMST (0033)	内蔵された時計をセットする。(時計はこのコールで起動される。) コールの条件は、 ACC←0 (AM), ACC←1 (PM) DE ← 時間を、秒に直したもの (バイナリで 2 バイト)	AF 以外は 保存	6
CALL TIMRD (003B)	内蔵された時計の値を読み取る。リターン時の状態は、 ACC←0 (AM), ACC←1 (PM) DE ← 時間を、秒に直したもの (バイナリで 2 バイト)	AF, DE 以 外は保存	3
CALL BRKEY (001E)	[SHIFT] + [BREAK] が押されたかどうかをチェックする。 押されていれば、Zフラグはセット。 押されていなければ、Zフラグはリセット。	AF 以外は 保存	1
CALL GETL (0003)	キーボードから 1 行分を入力する。 あらかじめ、入力データをストアするスタートアドレスを DE レジスタに指定しておくこと。エンドマークはキャリッジリタ ーンによる。(この場合モニタは SP-1001, SP-1002 いずれで あってもエンドマークとして 0D コードがセットされる。) 入力文字数は、0D コードを含めて最大 80 文字である。 キー入力にはエコーバックが行われ、カーソルコントロールも 受けつけられる。 [SHIFT] + [BREAK] が押されたら、DE レジスタの示すアド レスの先頭に BREAK コード、次にキャリッジリターンがセット されて戻る。	全レジスタ 保存	15
CALL GETKY (001B)	キーボードから 1 文字だけ ASCII コードを ACC に取り込む。 そのときキー入力がなければ、ACC に 00 がセットされて戻る。 但し、キー入力によるチャタリングは防止しておらず、エコー バックも行わない。 この GETKY を行う場合、モニタのバージョンによって、ACC に取り込まれる特殊キ ーコードが異なっている。[DEL] や [CR] などのキーがそれである。 これらの特殊キーの入力によって ACC に取り込まれるコードを次に示す。 ここで "ディスプレイコード" というのは、キャラクタジェネレータ内にあるキャラ クタを呼び出すためのコード番号である。実際に ACC に取り込まれるのはモニタプロ グラムのバージョンナンバによって異なる。	AF 以外は 保存	9
GETKY による特 殊キーの取り込み	ACC に取り込まれるコード		
	特 殊 キ ー	ディスプレイコード	

サブルーチン名 (16進番地)	サブルーチンの機能	レジスタ 保 存	スタック 数																												
CALL ASC (03DA)	ACCの下位4ビットを16進数と見做し、ASCIIに変換したものをACCにセットしてリターンする。	AF以外は保存	1																												
CALL HEX (03F9)	ACCの8ビットをASCIIと見做し、16進数に変換したものを、ACCの下位4ビットにセットしてリターンする。 リターン時のCF = " 0 " ACC←16進数 リターン時のCF = " 1 " ACCは保証されない。	AF以外は保存	1																												
CALL HLHEX (0410)	連続した4個のASCII列を16進数列と見做し、HLレジスタにセットしてリターンする。コール及びリターン条件は次の通り。 DE←ASCII列の先頭アドレス(例 " 3 " " 1 " " A " " 5 ") CALL HLHEX ↑ DE CF=0 HL←16進数(例 HL=31A5H) CF=1 HLは保証されない。	AF, HL以外は保存	2																												
CALL 2HEX (041F)	連続した2個のASCII列を16進数列と見做し、ACCにセットしてリターンする。コール及びリターン条件は次の通り。 DE←ASCII列の先頭アドレス(例 " 3 " " A ") CALL 2HEX ↑ DE CF=0 ACC←16進数(例 ACC=3AH) CF=1 ACCは保証されない。	AF, DE以外は保存	2																												
CALL ??KEY (09B3)	カーソルを点滅させながら、キー入力を待つ。キー入力があるとディスプレイコードに変換してACCにセット後、リターンする。	AF以外は保存	7																												
CALL ?ADCN (0BB9)	ASCII値をディスプレイコードに変換する。コール及びリターン条件は次の通り。 ACC←ASCII値 CALL ?ADCN ACC←ディスプレイコード	AF以外は保存	3																												
CALL ?DACN (0BCE)	ディスプレイコードをASCII値に変換する。コール及びリターン条件は次の通り。 ACC←ディスプレイコード CALL ?DACN ACC←ASCII値	AF以外は保存	3																												
CALL ?BLNK (0DA6)	テレビ画面の垂直ブランキングをチェックする。垂直ブランキング期間になるまで待っており、ブランキングになったらリターンする。	全レジスタ保存	2																												
CALL ?DPCT (0DDC)	テレビ画面上のディスプレイをコントロールする。コール時のACCとコントロールの関係は次の通り。 <table border="1"> <thead> <tr> <th>ACC</th><th>コントロール内容</th><th>ACC</th><th>コントロール内容</th></tr> </thead> <tbody> <tr> <td>C0H</td><td>スクローリング</td><td>C6H</td><td>[CLR] キーと同機能</td></tr> <tr> <td>C1H</td><td>⬇️ キーと同機能</td><td>C7H</td><td>[DEL] キーと同機能</td></tr> <tr> <td>C2H</td><td>⬆️ キーと同機能</td><td>C8H</td><td>[INST] キーと同機能</td></tr> <tr> <td>C3H</td><td>➡️ キーと同機能</td><td>C9H</td><td>[英数] キーと同機能</td></tr> <tr> <td>C4H</td><td>⬅️ キーと同機能</td><td>CAH</td><td>[カナ] キーと同機能</td></tr> <tr> <td>C5H</td><td>[HOME] キーと同機能</td><td>CDH</td><td>[CR] キーと同機能</td></tr> </tbody> </table>	ACC	コントロール内容	ACC	コントロール内容	C0H	スクローリング	C6H	[CLR] キーと同機能	C1H	⬇️ キーと同機能	C7H	[DEL] キーと同機能	C2H	⬆️ キーと同機能	C8H	[INST] キーと同機能	C3H	➡️ キーと同機能	C9H	[英数] キーと同機能	C4H	⬅️ キーと同機能	CAH	[カナ] キーと同機能	C5H	[HOME] キーと同機能	CDH	[CR] キーと同機能	全レジスタ保存	10
ACC	コントロール内容	ACC	コントロール内容																												
C0H	スクローリング	C6H	[CLR] キーと同機能																												
C1H	⬇️ キーと同機能	C7H	[DEL] キーと同機能																												
C2H	⬆️ キーと同機能	C8H	[INST] キーと同機能																												
C3H	➡️ キーと同機能	C9H	[英数] キーと同機能																												
C4H	⬅️ キーと同機能	CAH	[カナ] キーと同機能																												
C5H	[HOME] キーと同機能	CDH	[CR] キーと同機能																												
CALL ?PONT (0FB1)	現在のテレビ画面におけるカーソル位置をHLにセットする。リターン条件は次の通り。 CALL ?PONT HL←テレビ画面上のカーソル位置(バイナリ)	AF, HL以外は保存	5																												

7-3 ディスプレイコード表

ディスプレイコードとは、キャラクタジェネレータ内にあるキャラクタを呼び出すためのコード番号であり、テレビ画面へのキャラクタ表示はいつも、このコードをビデオRAMに転送することによって行なわれている。

モニタサブルーチンの"PRNT (0012H)"ルーチンや"MSG (0015H)"ルーチンでは、アスキーコードを、このディスプレイコードに変換して、カーソルの示すビデオRAMの位置へ、そのコードを転送しているのである。但し、表中のC1H~C6Hのコードはカーソルコントロール用の文字であり、カッコのついていないカナ文字、たとえば"ァ" (9EH)、などは小文字をあらわす。

DISPLAY

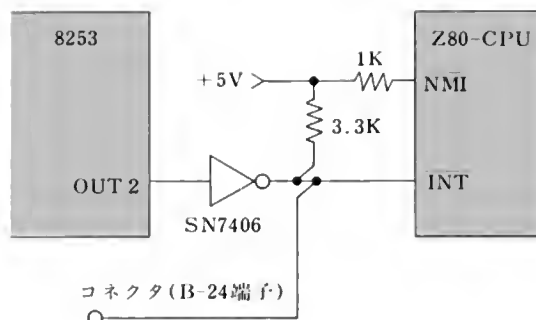
MSD LSD		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	SP	P	0	□	SP	↑	π	□	SP	セ	ワ	△	↓	日	□	SP
1	0001	A	Q	1	□	♠	◀	!	□	チ	夕	又	┘	↓	月	□	□
2	0010	B	R	2	□	◀	┌	"	□	コ	ス	フ	イ	↑	火	□	□
3	0011	C	S	3	□	□	♥	#	□	ソ	ト	ア	ユ	→	水	〜	□
4	0100	D	T	4	□	♦	┐	\$	□	シ	カ	ウ	ヲ	←	木	々	□
5	0101	E	U	5	□	←	@	%	□	イ	ナ	エ	、	H	金	示	□
6	0110	F	V	6	□	♣	◀	&	◀	ハ	ヒ	オ	ウ	C	土	入	□
7	0111	G	W	7	□	●	▶	!	◀	キ	テ	ヤ	ヨ	☼	生	ト	□
8	1000	H	X	8	□	◎	☐	(□	ク	サ	ユ	○	H	年	木	□
9	1001	I	Y	9	□	?	∖)	□	ニ	ン	ヨ	・	工	時	K	□
A	1010	J	Z	一	□	●	☐	+	□	マ	ツ	ホ	エ	大	分	K	□
B	1011	K	田	=	□	☐	☐	*	□	ノ	□	ハ	ツ	木	秒	入	□
C	1100	L	□	;	□	☐	☐	☐	□	リ	ケ	レ	ハ	木	円	ト	□
D	1101	M	□	/	□	☐	☐	☐	□	モ	「	メ	○	¥	¥	ト	□
E	1110	N	H	□	□	☐	☐	☐	□	ミ	ア	ル	オ	☺	☺	ト	□
F	1111	O	□	、	□	☐	☐	☐	□	ラ	ヤ	ネ	ー	☺	☺	☐	☐

7-4 割込みを使用する上での注意

MZ-80Kの本体後部のコネクタには、Z80-CPUのINT信号が接続されている。INT信号周りの回路は右図のようになる。外部からINT信号を接続する場合はオープンコレクタのドライバを利用するのが望ましい。

ノンマスクابل・インタラプト信号 (NMI) は内部でプルアップしてある。(プルアップ抵抗は1 K Ω)

またMZ-80Kのモニタプログラムでは割込みモードをモード1に設定し割込み禁止の状態にある。このあとの動作は次の2通りが考えられる。



(1) BASIC プログラムの TIS を動作させる場合

現在MZ-80K用として開発されているBASICプログラムはすべてTISによって時計機能を付与されている。計時用カウンタとして8253のカウンタ#1、#2共に利用しているが、BASICプログラムの起動時には0時0分0秒を設定している。8253のカウンタ設定と同時に割込みが許可される。その後、12時間ごとに8253のOUT2端子がハイレベルになり、INT信号による割込みでプログラム上の24時間時計を作っている。

このためBASICプログラム動作時に、外部からINT信号を使用するには十分な注意を必要とする。BASICプログラム内では割込みモードを1として時計機能を動作させているわけで、CPUが割込みを受けると、0038H番地へのリスタート命令を実行する。この0038H番地には"JP 1038H"命令が置かれ、更に1038H番地から時計の更新プログラムへジャンプしている。そこへ外部からINT信号を有効にすると、8253からの割込みと区別がつかなくなる。以上の理由から、BASICプログラム動作にはINT信号を使わないか、使う場合には時計機能を止めるかなければならない。

(2) ユーザー独自のプログラムを動作させる場合

ユーザーがSYSTEM PROGRAMを使って、独自のプログラミングを行なう場合INT信号は任意に利用することができる。Z80-CPUのマスカブルインタラプトは、モード0、1、2があり、モニタプログラムではモード1に設定してあるので、ユーザープログラム内で、任意のモードに変えて使用する。さらに8253にはリセット機能がないので、OUT2からの割込みをマスクするには、コントロールワードをセットするだけにして、カウンタのプリセットを実行しないようにすればリセットと同じ機能がはたらく。

またモニタプログラム内では割込み禁止にセットしてあるので、適宜"EI"命令を実行する必要がある。

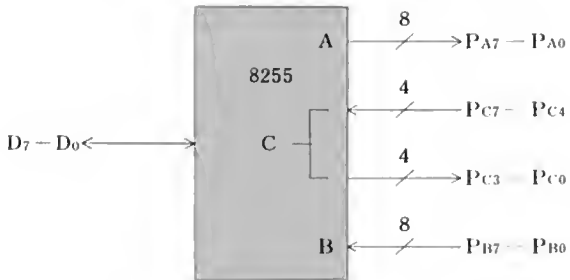
7 — 5 E000_H 番地内の考え方

E000_H 番地は各種端末コントロール用として、メモリマップド I-O (Memory Mapped I-O) を設定している。本体ボードの8255 (Programmable Peripheral Interface), 8253 (Programmable Interval Timer) に E000_H ~ E008_H 番地まで割り当てられる。

モニタプログラムにおける8255, 8253のモード設定は次のようになる。

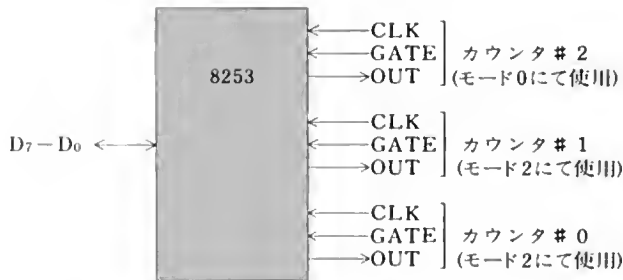
コントロールワード

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	1	0



コントロールワード

D7	D6	D5	D4	D3	D2	D1	D0	
0	1	1	1	0	1	0	0	カウンタ #2 に1秒定数をセット。
1	0	0	0	0	0	0	0	カウンタ #2 のリード。
0	0	1	1	0	1	0	0	分周比セット。



8255のポート定義 (モード 0)

A				B				グループ A				グループ B			
D4	D3	D2	D1	ポート A	ポート C (上4ビット)	#	ポート B	ポート C (下4ビット)							
0	0	0	0	出力	出力	0	出力	出力							
0	0	0	1	出力	出力	1	出力	入力							
0	0	1	0	出力	出力	2	入力	出力							
0	0	1	1	出力	出力	3	入力	入力							
0	1	0	0	出力	入力	4	出力	出力							
0	1	0	1	出力	入力	5	出力	入力							
0	1	1	0	出力	入力	6	入力	出力							
0	1	1	1	出力	入力	7	入力	入力							
1	0	0	0	入力	出力	8	出力	出力							
1	0	0	1	入力	出力	9	出力	入力							
1	0	1	0	入力	出力	10	入力	出力							
1	0	1	1	入力	出力	11	入力	入力							
1	1	0	0	入力	入力	12	出力	出力							
1	1	0	1	入力	入力	13	出力	入力							
1	1	1	0	入力	入力	14	入力	出力							
1	1	1	1	入力	入力	15	入力	入力							

8253のコントロールワードフォーマット

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RL1	RL0	M2	MI	M0	BCD

SC1	SC0	カウンタの選択
0	0	カウンタ・0
0	1	カウンタ・1
1	0	カウンタ・2
1	1	不適用

RL1	RL0	リード/ロード選択
0	0	カウンタの内容をラッチング
0	1	MSBだけのリード/ロード
1	0	LSBだけのリード/ロード
1	1	MSB, LSBの順のリード/ロード

M2	MI	M0	モード選択
0	0	0	モード 0
0	0	1	モード 1
×	1	0	モード 2
×	1	1	モード 3
1	0	0	モード 4
1	0	1	モード 5

BCD	Binary/BCD選択
0	16ビットバイナリカウンタ
1	4桁BCDカウンタ

7-6 メモリマップド I-O チャート

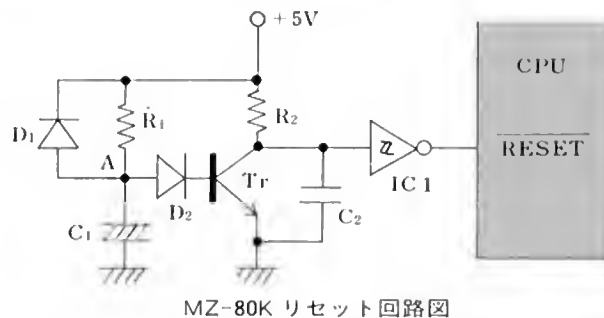
番地(16進)	メモリ・リード	メモリ・ライト
E000 (KEYPA)		D7 カーソル点滅用タイマのリセット D3 } } キーボードスキンのロー出力 D0 }
E001	D7 } } キーボードスキンのカラム入力 D0 }	
E002 (KEYPC)	D7 V-BLANK D6 カーソル点滅用タイマのステータスビット D5 カセット装置からの読み取りデータ D4 カセット装置のRecord/Playボタンの検出	D3 カセット装置のモータをON/OFFするためのパルス D2 英数/カナランプ D1 カセット装置への書き込みデータ D0 V-GATE
E003 (KANAST)		8255のCポートのシングルビットセット/リセット 英数(青)ランプ LD A, 05H LD (E003H), A カナ(赤)ランプ LD A, 04H LD (E003H), A
E004		8253のカウンタ・0のセット MZ-80Kでは2MHZのクロックを8253のカウンタ・0で分周して音階周波数を発生させている。モニタサブルーチンに於いて説明した、"CALL MSTA"で分周比のセットとGATEの制御がなされている。
E005	8253のカウンタ・1の読み取り	8253のカウンタ・1のプリセット
E006	8253のカウンタ・2の読み取り	8253のカウンタ・2のプリセット
E007		8253のモード設定
E008	D0 テンポ用タイマのステータスビット	D0 Music Start(1), STOP(0)

7-7 ハードウェアリセットの考え方

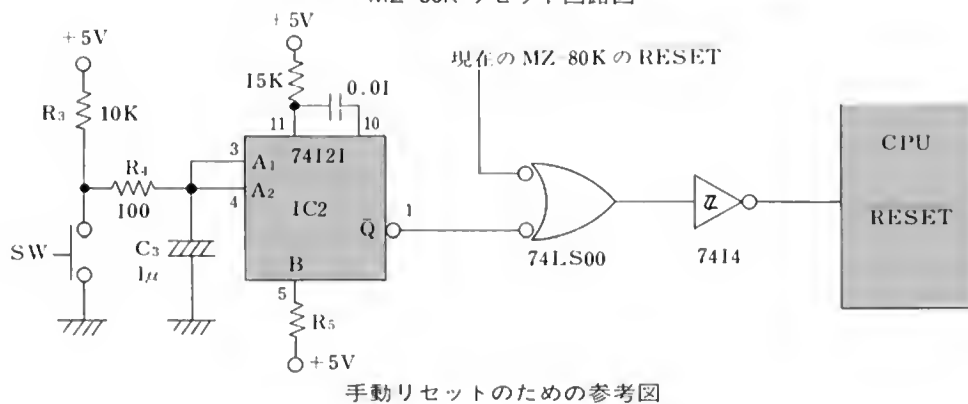
一般に、電源投入時におけるマイクロコンピュータの動作状態は定まっていない。定まった初期の動作開始状態に持ち込むために、リセット信号 RESET が用いられる。

リセット信号が与えられると、プログラムカウンタの内容が 0 になる。リセット信号に続いて、0 番地以降に書かれているモニタプログラムによって、MZ-80K の動作を開始させることが出来る。

MZ-80K では、電源投入と同時に、自動的にリセット信号が加わるように設計されている。その概略回路図を次に示す。



MZ-80K リセット回路図



手動リセットのための参考図

電源が ON となると、R₁ には C₁ の充電電流が流れ C₁ が充電してしまうまで、Tr のベース電位を低く保つ。この状態において Tr は OFF であるので、コレクタ出力は高電位となる。インバータ IC1 によって反転させられて、CPU の RESET は低電位となりリセットされる。

C₁ の充電によって、A 点の電位が Tr の V_{BE} と D₂ の順方向電圧 V_D の和になると、Tr が ON となりリセット信号が解除される。電源が OFF になると C₁ に充電されていた電荷は D₁ を通じて放電されるので、電源を OFF にした後すぐに ON してもリセット信号は正常に動作する。

このように、MZ-80K では電源投入時にのみリセットが加わるのであるが、電源を切らないでリセット信号を与えたい場合が時々生ずる。たとえば、BASIC プログラムにおいて誤った POKE 命令の使い方をした場合に BASIC インタプリタが壊されて、暴走状態となる。再度実行させるには、一度電源を切らなくてはならない。電源を切断することによって RAM に書き込まれているデータ・プログラムは消去され再実行まで、時間がかかることになる。

手動によってリセットを与えるための回路図を上右図に示しておく。

IC2 は、単安定マルチバイブレータであり、図のように B が高電位に保たれている。動作中において、C₃ は充電されているので、SW-ON によって、A₁ 及び A₂ は低電位に落ちる。その結果 Q には負のパルスが発生しリセット信号として働く。

(尚、CPU がリセットされても、プログラムが正常かどうかは別問題ですので注意が必要です。また、この図は参考図であり、回路の詳細について責務は負いかねます。)

7 — 8 参考文献

- | | | | |
|-------------|------|-----------------------------|-----------------|
| シャープ(株) | 1978 | マイクロコンピュータ Z80ハンドブック | エレクトロニクスダイジェスト社 |
| シャープ(株) | 1978 | マイクロコンピュータ Z80アプリケーションマニュアル | エレクトロニクスダイジェスト社 |
| W. バーデン, Jr | | | |
| 寺田浩詔監訳 | 1979 | Z80マイクロコンピュータ | 丸善 |

■ はじめて学ぶ人のために

- | | | | |
|---------|------|-----------------------|-----------------|
| シャープ(株) | 1978 | マイコン読本 | エレクトロニクスダイジェスト社 |
| 樹下行三著 | 1977 | マイクロコンピュータ〔I〕 基礎編 | エレクトロニクスダイジェスト社 |
| | | マイクロコンピュータ〔II〕 プログラム編 | エレクトロニクスダイジェスト社 |
| 浦 昭二著 | 1970 | アセンブリ言語 | 培風館 |
| 庄司 渉著 | 1979 | わかるアセンブラプログラミング | 電波新聞社 |

■ 本格的な学習のために

- | | | | |
|------------|------|------------------------------|------------|
| J.J.ドノバン 著 | 1974 | システムプログラム I、II | 日本コンピュータ協会 |
| D. グリス 著 | 1978 | コンパイラ作成の技法 | 日本コンピュータ協会 |
| N.ウィルト 著 | 1979 | アルゴリズム+データ構造=プログラム
PASCAL | 日本コンピュータ協会 |

mz-80k



mz-80k



シャープ株式会社

本社 〒545 大阪市阿倍野区長池町22番22号 電話 大阪(06) 621 1221(大代表)
産業機器事業本部 〒639 11 奈良県大和郡山市美濃庄町492番地 電話 大和郡山(07435)3 5521(大代表)

●お客様相談窓口およびシャープエンジニアリング

北海道 (011)642 4649 東北 (0222)96-4649 関越 (0286)35-1151
東京 (03)893-4649 北陸 (0762)49-4649 中部 (052)322 4649
近畿 (06)643-4649 中国 (082)874-4649 四国 (0878)33-4649
九州 (092)572-4649 沖縄 (0988)62-2231
